Understanding User Stories

Garm Lucassen

SIKS Dissertation Series No. 2017-44
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Understanding User Stories
ISBN: 978-90-393-6890-9

Understanding User Stories

Computational Linguistics in Agile Requirements Engineering

User Stories Doorgronden

Computationele Linguïstiek in Agile Requirements Engineering
(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de rector magnificus, prof.dr. G.J. van der Zwaan, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op donderdag 30 november 2017 des middags te 4.15 uur door

Garm Geurt Lucassen

geboren op 10 juni 1991 te Voorburg

Promotor: Prof. dr. S. Brinkkemper
Copromotor: Dr. F. Dalpiaz

People Talk

Is what I discovered during my first weeks of high school. A new classmate asked me: "Is it true you prefer to finish something quickly, rather than perfectly?" "*Of course*," I replied, "the reading corner is of much more interest to me than perfectly reciting all of Europe's rivers". Variations on this sentiment are an important theme of my time in high school: breaker on my phone instead of Latin grammar, NET SEND instead of English literature and beer instead of physical education.

The start of my academic career was not much different. Or so I thought, until I heard my father remark to a friend: "*I have never seen Garm work this hard for anything*". Something had definitely changed, although I would continue prioritizing the good life over good grades for quite some time. One month into my third year at Utrecht University, I attended a guest lecture that made me think 'this is it' for the very first time. A week later, I excitedly knocked on Sjaak's door and proclaimed my ambition to investigate software product management and startups for my bachelor thesis. After putting me in contact with startup founders in their network, Sjaak and Slinger's guidance pushed me to combine my natural pragmatism with something new: attention to detail. Together, we wrote a paper on visual business modeling techniques and submitted it to the International Conference on Software Business. It was my first foray into the world of academia, and I was already hooked.

Now, five years later, this book is the culmination of my scientific endeavors, bundling six of our best works. The journey leading up to this has been a humbling experience that allowed me to grow both professionally as well as personally. For one, I have developed a better appreciation for the wide variety of flavors in which humans come. If you are reading this, thanks for contributing your own unique flavor to my life. Some of you, I would like to especially thank for being great: Thanks, Gemma, for indulging my need for 'zelluf doen' and explaining the importance of enjoying life. Thanks, Miro, for sparking my interest in computers and teaching me that life is too short to wake up with regrets. Thanks, Okke, for looking up to me and becoming someone to look up to. Thanks, Oma Mia, for being my biggest fan – you are probably the only one who has read *all* of my work. Thanks, Tante Anja, for exemplifying that other people live life in a different gear and that's OK. Thanks, Sjaak, for giving me my first opportunity to shine and your continuing

support in maintaining that shine. Thanks, Fabiano, for the endless supply of incredibly intelligent constructive criticism – I am pretty certain that you will surpass the smartest Italian in RE in the near future. Thanks, Jan Martijn, for coming up with more ideas one could ever hope to investigate. Thanks, Slinger, for your unlimited positivity and encouraging feedback – I hope we get to collaborate again some day. Thanks, Govert-Jan, for being my polar opposite and softening my temper. Thanks, Brian, Didar, Eko, Ivor, Kevin, Marcel, Maxim, Marcel, Philipp, and all other collaborators for the teamwork on our papers. Thanks, Amir, Armel, Basak, Davide, Erik, Ian, Jaap, Kevin, Leo, Michiel, Marcela, Sergio, Sietse, Vincent, Wienand and all others who came and went for the shared laughs, trials and tribulations. Thanks, David, Erik, Gertjo, Klaas, Ronald, Otto and Wim for sharing your expertise with me and participants of the software product management course – my work wouldn't have been possible without you. Thanks, Gerald, Greg, Hans-Bernd, Hara, Peter and Samuel for tirelessly dedicating yourself to pushing for software product management excellence, I hope to put your thought-leadership into practice soon. Thanks, Floris, for taking a shot and creating exemplary user stories that we continue to use in research today. Thanks, Joost, for your let's just do it mentality – one day we'll have a sixth floor of our own. Thanks, Kim, Linsey, Vera and Willem, for always saying hello with three exclamation marks. Thanks, Ramon, for the amazing meals and putting up with my lazy, messy lifestyle. Thanks, Joren, for the engaging conversation at any hour of the day. Thanks, Jurrian, for your gleeful randomness. Thanks, Josien, for continuing to give me fashion advice after all these years. Thanks, Axel, for turning up the tunes. Thanks, Pim, for escaping our confines and going on adventures with me. Thanks, Wernard, for demonstrating how to appreciate the simple pleasures in life, while also being a baller sometimes. Thanks, Tom, for being solid as a rock, you'll probably turn into the philosopher's stone at some point. Thanks, Daan, Joost, Max, Richard, Robin and Roderik for sharing so much fantastic food, delicious drinks and lengthy laughs. Thanks, Stef, for pulling me out of my comfort zone. Thanks, Paul, for the lengthy life discussions. Thanks, Jan, for pushing me to be less serious. Let's stay young forever.
– Garm Lucassen

Oh, a sleeping drunkard
Up in Central Park,
And a lion-hunter
In the jungle dark,
And a Chinese dentist,
And a British queen–
All fit together
In the same machine.

Nice, nice, very nice;
Nice, nice, very nice;
Nice, nice, very nice–
So many different people
In the same device.
- Bokonon, Cat's Cradle, Kurt Vonnegut

Contents

1 Introduction **1**
1.1 Engineering User Story Requirements . . . 3
1.2 Research Questions . . . 13
1.3 Research Methods . . . 16
1.4 Dissertation outline . . . 19

2 The Use and Effectiveness of User Stories in Practice **23**
2.1 Introduction . . . 24
2.2 Study Design . . . 25
2.3 User Story Usage . . . 27
2.4 Perception of User Story Effectiveness . . . 30
2.5 Validity Threats . . . 38
2.6 Related Literature: User Stories, and Perception and Experiments in RE . . . 39
2.7 Discussion and Conclusion . . . 41

3 Improving Agile Requirements **43**
3.1 Introduction . . . 44
3.2 A Conceptual Model of User Stories . . . 45
3.3 User Story Quality . . . 49
3.4 The Automatic Quality User Story Artisan . . . 58
3.5 AQUSA Evaluation . . . 66
3.6 Enhancements: Towards AQUSA v2 . . . 73
3.7 Related Work . . . 75
3.8 Conclusion and Future Research . . . 79

4 Improving user story practice with the Grimm Method 81
4.1 Introduction . . . 82
4.2 Research method . . . 83
4.3 Results . . . 88
4.4 Related Literature . . . 97
4.5 Discussion and Outlook . . . 98

5 Extracting Conceptual Models 101
5.1 Introduction . . . 102
5.2 Baseline: The Grimm Method . . . 103
5.3 NLP Heuristics for User Story Analysis . . . 109
5.4 The Visual Narrator Tool . . . 114
5.5 Quantitative Evaluation: Accuracy . . . 120
5.6 Related Literature . . . 136
5.7 Conclusion and Future Work . . . 140

6 Visualizing User Story Requirements 143
6.1 Background . . . 144
6.2 Introduction . . . 146
6.3 Visualization Method for User Stories . . . 147
6.4 Prototype Demonstration . . . 153
6.5 Related Literature . . . 156
6.6 Discussion, Conclusion and Future Work . . . 158

7 Behavior Driven Requirements Traceability 161
7.1 Introduction . . . 162
7.2 Background . . . 163
7.3 Behavior-Driven Traceability . . . 169
7.4 BDT Change Impact Analysis . . . 173
7.5 Evaluating BDT and BDT CIA . . . 175
7.6 Discussion and Outlook . . . 184

8 Conclusion **185**
8.1 Limitations and Threats to Validity 195
8.2 Future Work 197
8.3 Reflections 199

Bibliography **200**

Published Work **227**

Summary **229**

Samenvatting **231**

Curriculum Vitae **233**

SIKS Dissertation Series **235**

From humble beginnings, grew the astonishing modern world in which we now live. We cannot conquer the mountains, but our railroads now run through them with ease. We cannot defeat the river, but we can bend it to our will and dam it for our own purposes.

We now live in a time of endless possibility.

– Dr. John W. Thackery, The Knick

Introduction

1

Popular culture increasingly depicts software creators as the heroes of our time. David Fincher's critically acclaimed movie *The Social Network* (2010) portrays Mark Zuckerberg as an omni-potent genius who single-handedly created Facebook and then turned it into a global Internet phenomenon with a little help of his friends. A little closer to home, Dutch journalist Alexander Klöpping frequently tells the story of fearless Internet entrepreneurs that somehow transform our lives for the better. While true to some extent, the majority of software you and I use for work and play is created with much less glamor. Every day, teams of extraordinary nerds and creative geniuses gather in offices across the globe to work on enabling you to complete your tax returns from the comfort of your couch. Boring, but necessary. Mike Judge's HBO television show *Silicon Valley* is one of the few to get this more right than wrong. In the first episode the protagonist accidentally makes a breakthrough in compression technology; suddenly he has to orchestrate the creation of a video streaming platform. Under his leadership, a team of idiosyncratic software developers stumble through hilarious disaster after hilarious disaster. Nevertheless, the team does somehow manage to demo a record-breaking prototype at an industry event.

To get anything done at all in the real world, teams like this need agreements on how to work. The field that studies all aspects of developing software while following systematic methods is called *software engineering* (IEEE, 2010). A broad field that consists of 15 interrelated disciplines such as software design, software testing and software development (IEEE et al., 2014). It all begins, however, with *requirements engineering*: the discipline that focuses on researching, documenting and analyzing what a piece of software should do (Kotonya and Sommerville, 1998). The central concept of this discipline is the *requirement*, which may refer to either something a stakeholder wants the software to do or to a textual representation of this wish (IEEE et al., 2014).

Traditionally speaking, both software engineering and requirements engineering try to specify everything as detailed and perfect as possible according to ISO standards set by the Institute of Electrical and Electronics Engineers (IEEE, 2011). Unfortunately, however, requirements change. And not without consequence. Much like how you might make last-minute changes

to your dinner plans when confronted with today's special, requirements frequently change when new information becomes available. All the work spent specifying the perfect requirements is then wasted.

The need for software engineering methods capable of adapting to change brought rise to the *"Manifesto for Agile Software Development"* (Beck et al., 2001): a collection of values and principles to guide collaborative creation of working software in changing environments. Since, up to 94% of software professionals around the world have successfully adopted agile methods such as Scrum, Kanban and XP (Version One, 2017). Many agile methods recommend capturing requirements with ***user stories***: the topic of this dissertation.

What exactly is a user story? Outside the world of software the term refers to a testimonial of a happy customer or a narrative describing how users benefit from a product. To software professionals, however, a user story is something entirely different: a short description of something a piece of software is supposed to do, told from the perspective of the person who desires the new capability. Originally, user stories were proposed as unstructured text similar to the then industry standard use cases (Beck, 1999) but restricted in size (Cohn, 2004). Nowadays, most user stories follow a strict, compact template that captures *who* it is for, *what* it expects from the system, and (optionally) *why* it is important (Wautelet et al., 2014). Although many different templates exist, 70% of practitioners use the Connextra template (Lucassen et al., 2016a):

Template:	"As a ⟨*type of* ***user***⟩ , I want ⟨***goal***⟩, [so that ⟨*some* ***reason***⟩]"
Example 1:	***As a*** visitor, ***I want to*** purchase an event ticket
Example 2:	***As an*** event organizer, ***I want to*** search for new events by favorited organizers, ***so that*** I know of events first
Example 3:	***As an*** event organizer, ***I want to*** receive an email when a contact form is submitted, ***so that*** I can respond to it

Thanks to their simplicity and connection to agile methods, software professionals around the globe are adopting user stories to capture requirements. In a recent survey, user story adoption was found to be nearly 50% (Kassab, 2015). Yet, despite substantial adoption in industry, academic studies on user stories were virtually non-existent at the start of this research. Indeed, little is known about how user stories are actually used in requirements engineering practice, and it is unclear whether contemporary requirements engineering

research can be successfully applied to user stories. Strange, as their concise and simple textual structure makes user stories promising for solving many requirements engineering problems that can take advantage of computational linguistics. With this in mind, this dissertation investigates the topic of user stories from the inside out. By reading this dissertation, you will learn more about why user stories are popular, how to support creating high-quality user stories, and, most importantly, how to help practitioners in fully achieving the title of this dissertation: *"Understanding User Stories"*.

The remainder of this introductory chapter outlines our approach to reaching an understanding of user stories. The subsequent Section 1.1 introduces the research domains that we investigate in this dissertation and highlights the relevance of our work. We then pose a main research question accompanied by six research questions in Section 1.2, which we answer by applying the research methods detailed in Section 1.3. This results in six scientific papers that make up this dissertation's content. Section 1.4 summarizes each paper.

1.1 ENGINEERING USER STORY REQUIREMENTS

The contrast of user stories' widespread adoption with the little academic attention it receives is the primary motivation for this dissertation. Although research interest is slowly growing, the available literature is still limited. Meanwhile, the growing popularity of agile development practices such as Scrum leads to a continuously increasing adoption of user stories (Wang et al., 2014; Kassab, 2015). Indeed, the growth in Google searches on user stories shows no sign of slowing, and now substantially exceeds the number of searches on *"requirements specification"* as shown in Figure 1.1.

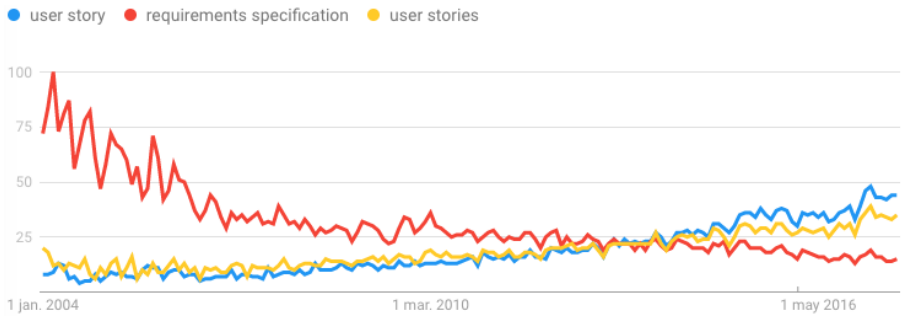


Figure 1.1: Google Trends data for searches on user story, user stories and requirements specification — retrieved on 25/08/2017

Another important motivation is user stories' focus on simplicity and practitioner's near-universal adoption of the Connextra template. Every day, industry practitioners formulate large number of user stories that are relatively uniform and not too complex. This predictable structure creates unique opportunities for applying computational linguistic techniques to user stories in order to solve long-standing requirements engineering challenges such as: (i) formulating high-quality requirements, (ii) creating models of the system's functionality and (iii) tracing a requirement to the source code that implements it (Nuseibeh and Easterbrook, 2000). Each of this dissertation's chapters presents an answer to one or more of these challenges by combining the state-of-the-art of multiple research domains. To aid the reader's understanding of these challenges and our proposed solutions, we first elaborate upon this dissertation's main research domain: requirements engineering. The subsequent subsections then introduce the four sub-domains this dissertation investigates:

I. Natural language processing
II. Requirements quality
III. Conceptual modeling and visualization
IV. Requirements traceability

1.1.1 *Requirements Engineering*

Requirements *engineering* or *RE* refers to capturing, refining and analyzing stakeholder wishes through the application of *systematic* methods (IEEE, 2011). In order for software developers to realize the *right* software system, they first need to understand *what* they're going to build and *why*. The notion that failing to establish this understanding is a bad idea is one of the fundamental theories of RE (Potts and Bruns, 1988; Lee and Lai, 1991; Yu and Mylopoulos, 1994). Thus, it is the requirements engineer's responsibility to continuously improve his understanding of why the stakeholders want what they want and to communicate this effectively to those who are actually going to build the software. A requirements engineer's typical day consists of activities such as: talking to customers, formulating requirements, considering which requirements have the highest priority and explaining requirements to other stakeholders (Pohl, 2010).

Note that requirements engineering is still a relatively young research domain. RE emerged as a self-sustaining, widely recognized discipline in the early 1990s, when two groups of software engineering researchers founded what is now the IEEE International Requirements Engineering Conference (Mead, 2013). Celebrating its 25th edition in 2017, the continuously growing body of work fueled by this conference has established RE as an important software

engineering discipline. Despite the wide variety of RE methods, approaches and supporting tools proposed during in the past 25 years, IT managers continue to blame incorrect requirements for software development failures and cost overruns (Standish Group, 2016). Indeed, there is considerable controversy regarding the effectiveness and necessity of RE.

On the one hand, there are numerous papers emphasizing some beneficial effect of RE. For example, Han and Huang find that the only differentiating factor among successful software projects is how they manage requirement risks, indicating that this is critical to achieving high software project performance (2007). Similarly, a large scale industry survey found a strong positive correlation between the success of large commercial application development and its organization's requirements management maturity (Ellis and Berry, 2013). To further improve requirements practice, the majority of improvement methods for RE propose a solution for one of three problems: ambiguity, incompleteness or inconsistencies (Pekar et al., 2014). Such a solution typically intends to prevent the introduction of errors in software development, which become exponentially costlier to correct in later phases (Westland, 2002).

On the other hand, there is initial empirical evidence that these problems and the RE activities to solve them are not as important as previously thought. Practitioner's perceived value of many of the practices for RE process improvement in Sawyer and Sommerville's requirements maturity model (Sawyer et al., 1997) is either low or none (Cox et al., 2009). Indeed, a study on the impact of requirement ambiguity for 40 industry projects was unable to detect (i) a correlation between requirement ambiguity and project success, nor (ii) that requirements ambiguity causes more defects during testing (Philippo et al., 2013). Similarly, while the IEEE discourages changing requirements in favor of creating complete requirements up front, Jørgensen found that projects whose requirements changed most often based on new information were more successful than projects with none or few requirements changes (Jørgensen, 2016). Moreover, a 2014 study found that experiment participants instructed to produce design concepts based on "ideas" rather than "requirements" came up with statistically significant more original designs (Mohanani et al., 2014).

Most aforementioned studies, however, suffer from a lack of rigor and replicability due to the inherent difficulty of collecting empirical data that unequivocally demonstrates problems or success factors in RE (Cheng and Atlee, 2007). Recognizing this problem, the *Naming the Pain in Requirements Engineering* (NAPIRE) initiative set out to identify contemporary RE problems according to 228 organizations across the globe (Méndez et al., 2016). They find that the primary problems in RE are (i) incomplete and/or hidden re-

quirements, (ii) communication flaws between project team and the customer, and (iii) moving targets. Coincidentally, these are the exact problems that the Agile Manifesto (Beck et al., 2001) and agile methods like Scrum try to mitigate by embracing user stories, the subject of this dissertation (Cohn, 2004). In Chapter 2 you will learn more about requirements engineering and user stories in particular. We explore how practitioners employ user stories in their day to day work and why they perceive them to be effective.

1.1.2 *Natural Language Processing*

Since the advent of the computer age computer scientists have been obsessed with teaching computers to understand human language. The holy grail of computing, the Turing test, tests a machine's intelligence by having human judges ask a series of questions to both a human and the machine. If the judges cannot identify which of the answers belong to the machine and which to the human, the machine has passed the Turing test and is considered 'intelligent' (Turing, 1950). The research field working toward beating this test is called Natural Language Processing or *NLP*.

Since the 1950s, NLP research has investigated many different approaches to reach this goal. Initial progress was made by meticulously designing rules for each problem. A notable example is the ELIZA chatbot, which mimics a psychotherapist by responding to most sentences with a variation of 'why do you say you ...' (Weizenbaum, 1966). While promising, these tools were unable to understand new idioms or domain specific jargon. As a response, many researchers started designing domain-specific ontologies to structure real-world information for computers to understand such as POLITICS (Carbonell, 1978). In many ways, chatbots like ELIZA and POLITICS are at the foundation of modern "intelligent" voice assistants like Apple's Siri and Amazon's Alexa. Ask Siri about ELIZA and she responds with "Eliza is my good friend. She was a brilliant psychiatrist, but she's retired now". Although contemporary voice assistants are much more advanced, posing a trick question quickly reveals Siri and Alexa are still far from intelligent.

Modern NLP research relies heavily on statistical models that apply a combination of methods and techniques to calculate the probability that a word is of a specific type. The model then assigns whichever type has the highest probability score (Manning and Schütze, 1999). Recent advances in NLP research that embrace co-occurrence matrices such as Word2Vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) substantially outperform the previous state-of-the-art and introduce innovative techniques such as "semantic

word arithmetics". Input "King" - "Man" + "Woman" and the computer will identify the female counterpart of King: "Queen".

Applying natural language processing to requirements engineering has historically been heralded as the final frontier of RE. However, unless a fundamental artificial intelligence breakthrough occurs, the ambitious objective of full RE automation is theoretically unattainable (Ryan, 1993; Berry et al., 2012). Therefore, RE studies apply NLP to specific use cases. Berry et al. (2012) categorize the fundamental approach of all NLP RE tools into four types:

1. Finding defects and deviations in NL requirements document;
2. Generating models from NL requirements descriptions;
3. Inferring trace links between NL requirements descriptions;
4. Identifying the key abstractions from NL documents

Despite significant advances in NLP RE research, the dozens of tools created for each type have failed to become widely adopted by industry. Berry et al. 2012 argue that these tools suffer from lack of adoption because of their inaccuracy. If the tool detects less than 100% of all issues, the analyst still has to repeat the entire task manually without any tool support. They propose—and we support his position—that tools that want to harness NLP should focus on the *clerical* parts of RE of which software has the potential of identifying all issues, leaving thinking-required work to human requirements engineers (Berry et al., 2012). For example, RE NLP tools should focus on detecting conjunctions such as *and*, *or*, and *+* or the use of an incorrect template, instead of difficult to detect potential inconsistencies such as the accidental use of the term *voucher* in one requirement, while all others mention *ticket*. Throughout this dissertation, we refer to this position as the *Perfect Recall Condition*.

User stories' focus on simplicity and the widespread adoption of the Connextra template causes user stories to exhibit characteristics of a *controlled natural language*: a subset of natural language with grammar and vocabulary restrictions that reduce ambiguity and complexity (O'Brien, 2003). While user stories do not enforce a limited vocabulary nor disallow any form of grammar, the guidelines and templates encourage formulating concise statements in the active voice. This structure makes user stories particularly well-suited for applying NLP techniques with the potential of achieving the Perfect Recall Condition. Chapters 3 through 7 all investigate this potential in some respect and present an NLP RE tool of Type 1, 2 or 3. For example, Chapter 3 presents specifically tailored NLP techniques for detecting a subset of the QUS Framework's criteria with the **A**utomatic **Q**uality **U**ser **S**tory **A**rtisan tool and Chapter 4 evaluates the impact of introducing the QUS Framework and AQUSA tool at three software companies.

1.1.3 *Requirements Quality*

Within the requirements engineering domain, a *requirement* is first and foremost something a stakeholder wants, but simultaneously also refers to something that captures a stakeholder's need for future reference. It is possible to express a requirement in many different ways, ranging from formal mathematical notations to rough sketches of a piece of technology (Fraser et al., 1994; Goel, 1991). In reality, however, the vast majority of requirements are captured using words that form human-readable sentences: natural language (Kassab, 2015). Much like other textual material, such as news articles, Whatsapp messages or dissertations, a requirement can be done well or poorly. Fueled by the notion that poor requirements lead to wrong software and unnecessary rework (Davis, 1993), many requirements engineering researchers have sought to determine what makes a good requirement good.

This has resulted in a wide variety of models and frameworks that prescribe attributes, characteristics or properties that a requirement should adhere to (Saavedra et al., 2013). The IEEE 29148 Standard *Systems and software engineering—Life cycle processes—Requirements engineering* (Table 1.1) is the standard work on this subject, defining characteristics that each stakeholder, system and system element requirement shall possess: 9 for a single requirement, 4 for requirements sets and 9 language criteria (IEEE, 2011).

Single	**Sets**	**Language**
Necessary	Complete	Superlatives
Implementation free	Consistent	Subjective language
Unambiguous	Affordable	Vague pronouns
Consistent	Bounded	Ambiguous adverbs and adjectives
Complete		Open-ended, non-verifiable terms
Singular		Comparative phrases
Feasible		Loopholes
Traceable		Incomplete references
Verifiable		Negative statements

Table 1.1: Requirement characteristics of the IEEE 29148 standard

Examples are (i) Singular - include "only one requirement with no use of conjunctions", (ii) Affordable - "solution is obtainable/feasible within life cycle constraints" and (iii) avoid comparative phrases such as 'better than', 'higher quality'. Unfortunately, most requirements specifications are unable to adhere to them in practice (Glinz, 2000), even though some authors found that high-quality requirements correlate with project success (Kamata and Tamai,

2007). Recognizing this, several authors have proposed alternative yet similar requirements quality models such as the Génova et al. quality model (2013) that defines fifteen measurable indicators that influence 11 desirable properties of a requirement and Fabbrini et al's quality model (2001) that emphasizes four high level quality properties of natural language requirements: testability, completeness, understandability and consistency.

Despite this large body of work on general requirements quality, the number of quality frameworks to assess and improve *user story* quality is limited. Existing approaches either (i) employ highly qualitative metrics, such as the six mnemonic heuristics of the *INVEST* (Independent–Negotiable–Valuable–Estimable–Scalable–Testable) framework by Bill Wake (2003), or (ii) present generic guidelines for quality in agile RE, such as the *Agile Requirements Verification Framework* by Petra Heck and Andy Zaidman (2014) which includes just three criteria specific to user stories. To bridge this gap, Chapter 3 investigates the notion of requirements quality in user story context. In this chapter, we introduce the **Q**uality **U**ser **S**tory Framework, which proposes 13 criteria to determine user story quality based on a literature review and critical analysis of hundreds of user stories.

1.1.4 *Conceptual Modeling and Visualization*

Natural language requirements are popular because they are both easy to create and easy to understand as they use the very same language that we use to communicate with other humans. Yet, natural language requirements also suffer from several drawbacks. Most importantly, words and sentences' inherent ambiguity frequently results in different interpretations of the same requirements text (Berry et al., 2001). Secondly, it is difficult to identify and explore the key entities and relationships in a large set of requirements. This makes it easy to overlook a dependency or conflict between the fifth and forty-fifth user story you read.

The recommended technique to prevent this from happening is to derive conceptual models from a natural language requirements collection (Sagar and Abirami, 2014). For requirements, a conceptual model typically includes all relevant entities and their relationships like in Figure 1.2. Although this model lacks the level of detail of a textual requirement, it effectively highlights the relationships of the requirements' most important entities.

Several studies have demonstrated the added value of conceptual modeling in software development (Davies et al., 2006; Fettke, 2009). Yet, practitioners remain reluctant to adopt these methods. In 2006, just 13% of the Australian

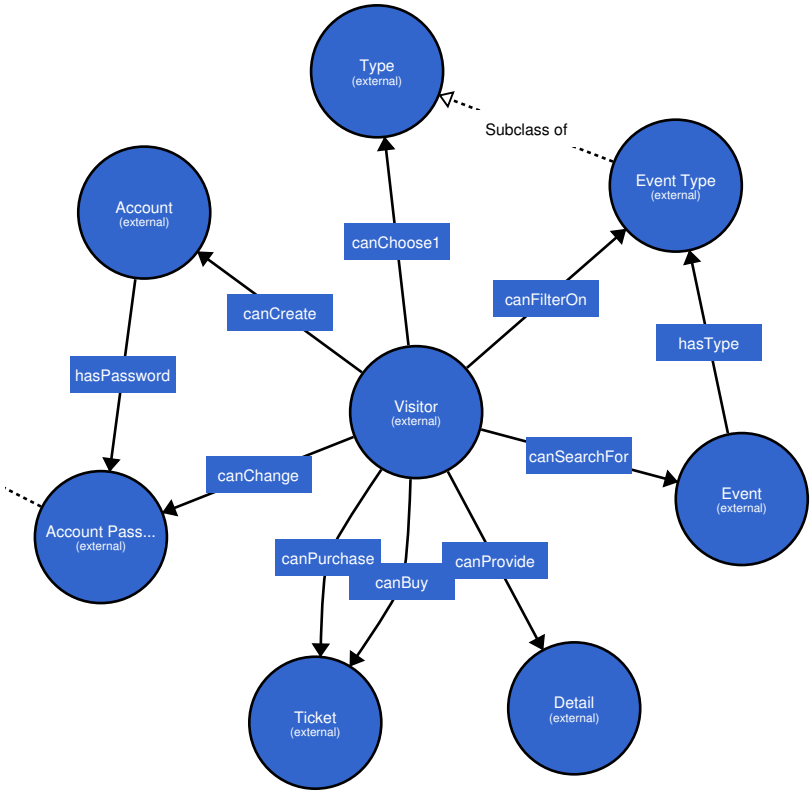


Figure 1.2: Example conceptual model for an event ticketing website visitor in the WebVOWL tool by Lohmann et al. (2015)

Computer Society's 12,000 members indicated an interest in conceptual modeling (Davies et al., 2006). A probable explanation is that the benefits do not outweigh the burden of manually constructing and maintaining a conceptual model. Consequentially, many authors have proposed tools to support a requirements engineer in constructing conceptual models (Saeki et al., 1989; Du and Metzler, 2006; Sagar and Abirami, 2014). All these tools, however, either (i) *require human supervision* to appropriately tag the entities and relationships in the text, or (ii) *have low accuracy*, often due to the ambitious attempt to support arbitrarily complex requirements statements.

Recognizing that this situation hampers industry adoption, Yue et al. called for future approaches that fully automatically generate complete, consistent and correct models (Yue et al., 2010). Again, user stories' focus on simplicity and concise structure creates an opportunity to surpass the current state-of-the-art. Chapter 5 investigates how to apply well-known conceptual modeling heuristics to user stories and demonstrates that our *Visual Narrator* tool is capable of fully automatically constructing conceptual models.

However, Visual Narrator's resulting models quickly become too large to be effectively explored by analysts: humans' working-memory capacity restricts the ability to read models and leads to *cognitive overload* when the same model includes too many concepts (Aranda et al., 2007; Moody, 2009). In Chapter 6 we overcome this problem by building upon Shneiderman's *visual information seeking* mantra: *"overview first, zoom/filter, details on demand"* (Shneiderman, 1996). We propose and experiment with a mechanism for improving Visual Narrator's output by applying clustering techniques that take advantage of state of the art semantic relatedness algorithms (Goldberg and Levy, 2014; Trask et al., 2015).

1.1.5 *Requirements Traceability*

At some point in time, software developers start realizing a requirement by creating software that reflects the stakeholder's wish. They produce so-called *source code*: a set of instructions that command a computer to perform specific actions. The field that tries to figure out why and how to link a requirement to the source code that realizes it, is called requirements traceability (Cleland-Huang et al., 2014a). Benefits of having this connection available include faster software development and increased program comprehension (Mäder and Egyed, 2015). Despite this positive influence, the majority of practitioners do not adopt traceability practices in their projects (Blaauboer et al., 2007). Two main reasons are that (i) the stakeholders who need to create the traces are not the ones who reap the benefits, the so-called traceability benefit problem (Arkley and Riddle, 2005), and (ii) the lack of methods and tools supporting traceability (Bouillon et al., 2013) hamper the effective adoption of traceability practices (Cleland-Huang et al., 2014a).

Recognizing these problems, the Center of Excellence for Software and Systems Traceability put forward the grand challenge of "always there" *ubiquitous traceability* that is "built into the software engineering process" (Cleland-Huang et al., 2014a). This has resulted in a variety of approaches and tools that take advantage of artifacts produced during existing software development

processes such as design documents and work completion messages (Borg et al., 2014; Duarte et al., 2016). Many rely on information retrieval algorithms to identify source code that is likely to correspond to a specific requirement or the other way around, similar to how search engines like Google guess which web pages are relevant for a given word sequence. By design, the performance of these algorithms is highly dependent on the input data quality and data type (Merten et al., 2016). Although the employed information retrieval algorithms are very promising, they are still far removed from the 100% accuracy necessary for ubiquitous traceability.

In Chapter 7, we present the automated Behavior Driven Traceability method to trace a user story to the source code that implements it by taking advantage of automated acceptance tests. Created as part of the Behavior Driven Development process or BDD (North, 2006), these tests are typically captured using the Gherkin syntax:

Template:	"***given*** some initial context, ***when*** an event occurs, ***then*** ensure some outcome"
Example:	***Given*** I go to the contact page ***When*** I submit a question ***Then*** the administrator receives a message

In this way, automated acceptance tests refine user stories into small steps that mirror a user's interaction with the system. Consequentially, we can establish ubiquitous traceability from a user story to the source code executed as part of these tests without requiring the assistance of imprecise algorithms.

1.2 RESEARCH QUESTIONS

The few studies available at the start of this research propose methods and tools to support or improve some aspect of user story practice (Rees, 2002; Miranda et al., 2009; Mahnič and Hovelja, 2012). Little is known about how user stories are actually used in RE practice, and it is unclear whether contemporary requirements engineering research can be successfully applied to user stories. A missed opportunity, as user stories' straightforward format and focus on simplicity makes them ideal candidates for applying natural language processing. We hypothesize that these characteristics create a unique opportunity to develop highly accurate computational linguistic techniques that support creating, communicating and discussing user stories. Thus this dissertation poses the following main research question:

MRQ —*How to employ computational linguistic techniques to understand user story requirements?*

Although user stories were already popular at the time this research started, the requirements engineering community has not yet scientifically established a rigorous baseline for user stories. High profile agile requirements engineering articles such as (Paetsch et al., 2003; Cao and Ramesh, 2008) do not go beyond summarizing the books that introduced user stories (Beck, 1999; Cohn, 2004), while mainstream requirements engineering books still fail to mention user stories (Pohl, 2010; Hull et al., 2010).

Recognizing this shortcoming, this dissertation first poses three research questions that seek to establish a clear user story quality baseline by carefully analyzing current industry practice. Research questions four through six build upon this foundation to explore processing a user story collection for increasingly advanced analyses. The subsequent paragraphs introduce each research question's motivation and our approach to answering it.

RQ1 — *How do industry practitioners use and perceive user stories?*
The first step towards answering our main research question is to gain a deeper understanding of what a user story is, exactly. Aside from the descriptive book by Mike Cohn "User Stories Applied: For Agile Software Development" (Cohn, 2004), there is an abundance of informal literature promising all sorts of positive effects by adoption user stories in some way (Jeffries; Patton and Economy, 2014). No literature is available, however, on how software professionals actually work with user stories. Furthermore, while the increasing popularity

of user stories suggests that practitioners consider them effective for capturing requirements, no scientific evidence for or against this is available. This research question aims to a) identify the context of user stories by looking at how practitioners approach working with user stories and b) examine whether practitioners agree that user stories increase their work productivity and/or deliverable quality.

RQ2 — *What are the textual characteristics of high-quality user stories?*
Although there is a large body of work on quality frameworks for requirements in general (Saavedra et al., 2013; IEEE, 2011), the number of quality frameworks to assess and improve *user story* quality is limited. Existing approaches either (i) employ highly qualitative metrics (Wake, 2003), or (ii) present generic guidelines for quality in agile RE (Heck and Zaidman, 2014). As a consequence, practitioners still struggle to create high quality user stories. With this research question we aim to learn what makes a good user story good and to define quality characteristics that practitioners can use to improve their own user stories.

RQ3 — *How can natural language processing support formulating high-quality user stories?*
Unfortunately, today's artificial intelligence is theoretically incapable of understanding a requirement's text with 100% accuracy (Ryan, 1993). However, thanks to user stories' concise format and focus on simplicity, there are some *clerical* parts of assuring a user story's quality which software *might* be able to identify with adherance to the Perfect Recall Condition as defined in Chapter 1.1.2. This research question aims to investigate how to employ computational linguistic techniques such as natural language processing to automatically detect defects within individual user stories and a set of user stories.

RQ4 — *How to identify principal concepts from a user story collection?*
A user story consists of three elements: *who* it is for, *what* it expects from the system, and (optionally) *why* it is important (Wautelet et al., 2014). Although many different templates exist, the de-facto standard with 70% adoption is the Connextra template (Lucassen et al., 2016a): *"As a ⟨type of user⟩ , I want ⟨goal⟩, [so that ⟨some reason⟩]"*. Every user story typically includes an actor that conducts an action on some object. While simple, when presented with a long list of user stories, it remains difficult to quickly identify and explore the roles, objects and actions amongst them. To obtain a holistic overview,

requirements engineers can organize a user story's entities and their relationships in conceptual models (Sagar and Abirami, 2014). Due to the steep time investment, however, adoption of methods to do this is low (Davies et al., 2006). This research question seeks to establish whether it is possible to apply computational linguistic techniques to user story's straightforward structure to automatically extract the most important entities and their relationships.

RQ5 — *How to visually organize the concepts of a user story collection?*
A well known drawback of conceptuals models is that they quickly become too large for humans to understand due to cognitive overload (Aranda et al., 2007). To reduce and structure the amount of information presented at once, a wide range of approaches is available (Moody, 2009). It is unclear, however, which of these are applicable for conceptual user story models. This research question aims to examine different visualization techniques that allow a human requirements engineer to quickly explore a user story collection.

RQ6 — *How to trace a user story to the source code it executes?*
The previous research questions all investigate user stories themselves, without taking into account their context. They are, however, part of the agile software development lifecycle and should lead to the creation of a piece of software. To increase program comprehension and software development speed it is useful to keep track of which software realizes which user story (Mäder and Egyed, 2015). Despite these benefits, practitioners refuse to adopt traceability practices due the traceability benefit problem (Arkley and Riddle, 2005) and the lack of tool support (Bouillon et al., 2013). This research question explores opportunities for establishing a trace between a user story and the source code that implements such as by using artifacts and data created as part of the software engineering process.

1.3 RESEARCH METHODS

In Chapter 2 through 7 we make use of many different research methods to formulate answers for the previous section's research questions. In the paragraphs below we briefly introduce the most relevant ones: surveys, case studies and design science. Furthermore, Table 1.2 gives an overview of the research methods applied in each chapter.

Design science is a research method to guide the creation of "things that serve human purposes" (March and Smith, 1995). Design science consists of two basic activities, build and evaluate, which we approach according to the framework and its specifications as proposed by Hevner and Chatterjee. This framework has been thoroughly tested as a vehicle for "understanding, executing and evaluating IS research combining behavioral-science and design-science paradigms". Although the work presented in this thesis does not always follow this framework's steps exactly, our work always includes a (preliminary) empirical evaluation of the things we built.

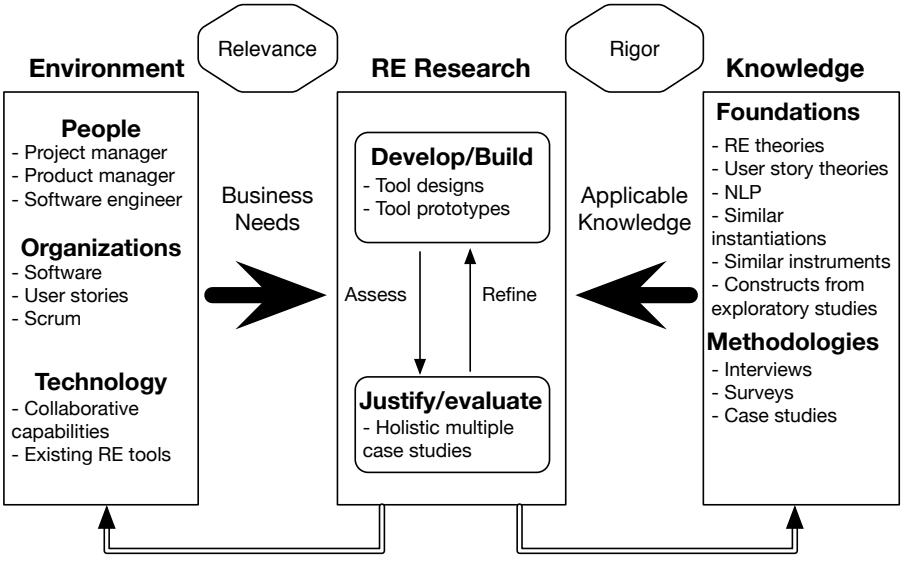


Figure 1.3: Hevner's Design Science Research Method Framework applied to this dissertation's research context

Figure 1.3 applies the framework to this dissertation's research context. We conduct this research in an **environment** consisting of *people* that perform the project manager, product manager or software engineer role in software *organizations*. All three roles capture requirements in *user stories* and communicate about them during *Scrum* processes. They would like to enhance their communicative *capabilities* by improving or extending existing requirements engineering *tools*.

The **knowledge base** *foundations* consist of existing *theories* on requirements engineering and user stories, NLP state-of-the-art, contemporary, similar *instantiations* and *instruments* as well as *constructs* developed as part of an initial exploratory study. The environment and knowledge base generate business needs and applicable knowledge inputs for our design science **RE research**, which *develops* two types of artifacts: tool designs and tool prototypes. Upon delivery, we *evaluate* the tool prototype in holistic multiple-case studies by leveraging interviews, surveys and case studies as *methodology*. The evaluation is input for an iterative process to refine the tool's design and prototype while taking into account any changes in the environment and knowledge base. This repeats until the tool is considered satisfactory.

Case studies are an important research method for obtaining empirical results in industry. The handbook for case study research is written by Yin 2009, who defines a case study as "an empirical enquiry that investigates a contemporary phenomenon in depth and within its real-life context". Note that the researcher is free to conduct the empirical enquiry in any way, as long as it takes place within a realistic scope. During a case study, participant(s) might for example apply a prototypical tool to their own data and then critically evaluate both the output and the usability of the tool using **interviews**. Yin distinguishes four types of case study designs according to holistic versus embedded and single versus multiple (Yin, 2009). For this dissertation, we exclusively make use of holistic multiple case designs: examining multiple companies' cases within their own context to learn more about one specific unit of analysis. The impact of introducing a user story quality framework, for example.

Surveys are treacherous tools when conducting research. While it is incredibly useful for obtaining large quantities of data, it is very easy to introduce flaws in your own survey design. Many books have been written on how to create strong surveys as well as on why a specific way of asking a question might be wrong or right. Our survey makes heavy use of Likert-Type questions, which we formulated and analyzed using the guidelines of Boone and Boone (Boone and Boone, 2012). In software engineering research, surveys are appropriate for three scenario's: (i) as a preliminary study before a more thorough investigation, (ii) making explanatory claims about a specific population, or (iii) establishing assertions about a specific population (Wohlin et al., 2012). The first is primarily useful in an exploratory stage whereas the latter two are reserved for rigorous evaluations.

1.4 DISSERTATION OUTLINE

Chapters 2 to 7 each report on our investigations into one or more of the research questions presented in Section 1.2. Table 1.2 gives an overview of the research questions, research methods and contributions of each chapter.

Chapter 1 — The *Introduction* puts this dissertation into context and highlights the relevance of this work. After a general introduction of user stories, it introduces the five research domains that the remaining chapters are going to investigate. Next, this chapter poses the research questions we intend to answer in this work and discusses the research methods we use to do this. It concludes with the dissertation outline you are reading right now.

Table 1.2: Research questions, research methods and contributions per chapter

Ch.	RQ	Research Method	Contributions
2	1	Survey	• User story use • User story effectiveness
3	2,3	Design science Case study research	• QUS framework • AQUSA tool
4	3	Survey Case study research	• Empirical QUS + AQUSA evaluation
5	4	Design science Case study research	• Visual Narrator • Grimm User Stories
6	5	Design science	• Interactive Narrator
7	6	Design science Case study research	• Behavior Driven Traceability

Chapter 2 — *The Use and Effectiveness of User Stories* explores how practitioners currently work with user stories by analyzing 182 survey responses and 21 follow-up interviews. The purpose is to go beyond anecdotal knowledge and gather scientifically grounded data on the use and perception of user stories in industry. Our results show that 70% of practitioners express user stories in the same format, the majority of practitioners agree that user stories contribute to work deliverable quality and productivity, and that those who employ quality guidelines are more positive about the benefits of user stories. This chapter has been published as a full research paper in the proceedings of the International Working Conference on Requirements Engineering: Foundations for Software Quality (REFSQ) (Lucassen et al., 2016a).

Chapter 3 — *The Quality User Story Framework and Tool* proposes the Quality User Story (QUS) framework, a set of 13 quality criteria that user story writers should strive to conform to. Based on QUS, the Automatic Quality User Story Artisan (AQUSA) software tool relies on natural language processing techniques to detect quality defects and suggest possible remedies. We describe the architecture of AQUSA, its implementation, and report on an evaluation that analyzes 1023 user stories obtained from 18 software companies. We find that between them, these user story collections contain 706 true errors, which we detect with 97.9% recall and 84.8% precision. This chapter has been published in the Requirements Engineering Journal (Lucassen et al., 2016b) as an extension of an earlier paper that was selected as a distinguished full research paper published in the proceedings of the International Requirements Engineering Conference (RE) (Lucassen et al., 2015).

Chapter 4 — *Improving User Story Practice With the Grimm Method* studies the effects of introducing the Grimm Method's Quality User Story framework and the AQUSA tool on the productivity and work deliverable quality of 30 practitioners from 3 companies over a period of 2 months. Despite an improvement in intrinsic user story quality, practitioners did not perceive this change. In follow-up interviews, some participants explain they did perceive an increase in constructive user story conversation leading to less rework. This chapter has been published as a full research paper in the proceedings of the International Working Conference on Requirements Engineering: Foundations for Software Quality (REFSQ) (Lucassen et al., 2017b).

Chapter 5 — *Extracting Conceptual Models from User Stories* extends and modifies existing natural language processing heuristics for extracting conceptual models. Our proposed tool Visual Narrator provides requirements engineers with a holistic overview of the system without the need for any human effort, allowing easy detection of requirements ambiguity and conflicts. By taking advantage of user stories' focus on the essential components of a requirement *who? what? why?*, we achieve state-of-the-art accuracy between 88% recall and 91% precision up to 98% recall and 97% precision.

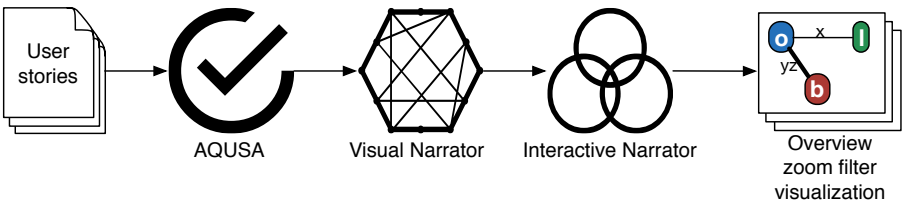


Figure 1.4: Simplified version of the Grimm User Story Method for agile requirements engineering

Furthermore, this chapter proposes our holistic approach to agile requirements engineering with user stories: the Grimm User Story Method. Figure 1.4 presents a simplified overview of how Grimm generates specialized visualizations from a user story collection by connecting our three RE NLP tool contributions: AQUSA, Visual Narrator and Interactive Narrator. This chapter has been published in the Requirements Engineering Journal (Lucassen et al., 2017c) as an extension of an earlier paper that was selected as a distinguished full research paper published in the proceedings of the International Requirements Engineering Conference (RE) (Robeer et al., 2016).

Chapter 6 — *Visualizing User Stories via Semantic Relatedness* builds upon the Visual Narrator tool in order to reduce the complexity of its initial output. The proposed solution combines state-of-the-art semantic relatedness algorithms with clustering techniques to present a proof-of-concept tool that creates overviews of clustered user story concepts and generates zoom views for each cluster to enable investigating the relationships of their underlying concepts. This chapter has been published as a full research paper at the International Conference on Conceptual Modeling (ER) (Lucassen et al., 2016c).

Chapter 7 — *Behavior Driven Requirements Traceability* proposes an approach to the automated traceability problem. Unlike established literature, we establish ubiquitous traceability between user stories and source code by taking advantage of the automated acceptance tests that are created as part of the Behavior Driven Development process. Furthermore, we demonstrate how to combine BDT output with the entity extraction tool Visual Narrator to perform change impact analysis when new user story requirements are added. This chapter has been submitted for journal publication (Lucassen et al., 2017).

Chapter 8 — The *Conclusion* gives an overview of the answers to all research questions, draws six conclusions based on the answers, highlights the scientific contributions of this work and presents four opportunities for future work.

The Use and Effectiveness of User Stories in Practice

2

[**Context and motivation**] User stories are an increasingly popular textual notation to capture requirements in agile software development. [**Question/Problem**] To date there is no scientific evidence on the effectiveness of user stories. The goal of this paper is to explore how practicioners perceive this artifact in the context of requirements engineering. [**Principal ideas/results**] We explore perceived effectiveness of user stories by reporting on a survey with 182 responses from practitioners and 21 follow-up semi-structured interviews. The data shows that practitioners agree that using user stories, a user story template and quality guidelines such as the INVEST mnemonic improve their productivity and the quality of their work deliverables.

[**Contribution**] By combining the survey data with 21 semi-structured follow-up interviews, we present 12 findings on the usage and perception of user stories by practitioners that employ user stories in their everyday work environment.

This work was originally published as:

G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. The Use and Effectiveness of User Stories in Practice. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 205–222, 2016a

2.1 INTRODUCTION

User stories (Cohn, 2004) are a popular method for representing requirements using a simple template such as *"As a ⟨role⟩ , I want ⟨goal⟩, [so that ⟨benefit⟩]"*. Their adoption is growing (Kassab, 2015), and is massive especially in the context of agile software development (Wang et al., 2014). Despite their popularity, the requirements engineering (RE) community has devoted limited attention to user stories both in terms of improving their quality (Lucassen et al., 2015) and of empirical studies on their use and effectiveness.

The purpose of this study is to go beyond anecdotal knowledge and gather scientifically rigorous data on the use and perception of user stories in industry. This includes data on the development methods they are used in, the templates used for structuring user stories, and the existing quality guidelines. Additionally, we explore whether practitioners perceive an added value from the use of user stories: Do they increase productivity? Do they ameliorate work deliverable quality?

Earlier studies have shown that RE practices play a central role in agile development (Hoda et al., 2010; Wang et al., 2014) albeit on a small scale and in a local context. Ramesh, Cao and Baskerville pinpointed agile RE practices and challenges by studying 16 organizations (Ramesh et al., 2010) but they have not studied the role of user stories in detail. Other works studied the effectiveness of RE practice and artifacts through experiments (Cruz-Lemus et al., 2007; Dieste and Juristo, 2011; Condori-Fernandez et al., 2009; Penzenstadler et al., 2013) as well as the use and perception of practitioners (Hofmann and Lehner, 2001; Abrahão et al., 2011).

This paper describes our conducted empirical research, which includes an online survey followed by semi-structured interviews with a subset of the survey respondents. Key findings of our analysis include the strong link between Scrum and user stories, the widespread adoption of the user story template proposed by Connextra, the perception that user stories help practitioners define the *right* requirements, the crucial role of explaining *why* a requirement is expressed, and a positive evaluation of quality frameworks by respondents that use one.

The remainder of this paper is structured as follows. Sec. 2.2 presents our research questions and describes the design of our empirical study. Sec. 2.3 analyzes the survey and interview results concerning the *use* of user stories in practice, while Sec. 2.4 reports on the *perceived effectiveness*. Sec. 2.5 discusses validity threats to our research, while Sec. 2.6 reports on related literature. We discuss our results and conclude in Sec. 2.7.

2.2 STUDY DESIGN

The goal of this study is to understand how practitioners use and perceive user stories, which prompts us to formulate two research questions:

RQ_1: How do practitioners use user stories?
We investigate the context of user stories by looking at how practitioners approach working with user stories. What software development methods are appropriate for using user stories? Which templates and quality guidelines are popular among practitioners?

RQ_2: How do practitioners perceive the effectiveness of user stories?
In this study, we decompose effectiveness into productivity and quality of work deliverables; although many more aspects exist, these are two basic performance indicators for software development processes. We examine whether practitioners agree that user stories increase their work productivity and/or the quality of their work deliverables. Additionally, we investigate whether practitioners find that utilizing a template and/or a quality framework further improves these aspects.

To answer these research questions, we split our study design in two stages: (1) we conduct an online survey that we distribute worldwide among software professionals to collect quantitative information from practitioners on the use of user stories and their added value for RE, and (2) we perform follow-up interviews to gather clarification of the answers of a selected sample of survey respondents, improving our understanding of the survey findings.

The authors distributed the survey over a variety of channels including the professional network of the authors and online communities such as requirements engineering and software engineering mailing lists, Twitter, Hacker News and Reddit Agile. Over a span of two weeks, from July 7 2015 until July 21 2015, the survey obtained 197 responses. 49 survey respondents were invited to participate in a follow-up interview, 21 of which accepted and contributed with more in-depth, qualitative data on the subject.

We analyzed the survey responses using SPSS, Excel and R; we transcribed the follow-up interviews and categorized them using the qualitative data analysis tool Nvivo.

2.2.1 *Research Protocol*

The goal of the survey is to gather quantitative data on how practitioners use and perceive user stories. To achieve this goal, we formulated 21 questions that are available in our online appendix (Lucassen, 2015). After a short introduction on our research, the survey asked five questions on the respondent's demographics and organizational context, followed by six questions on their usage of and experience with user stories, templates and quality guidelines. Next, respondents were asked to indicate whether they agree or disagree with the following six Likert-Type statements, which we reference by their number throughout the paper:

- **S1** Using user stories increases my productivity
- **S2** Using user stories increases the quality of my work deliverables
- **S3** Using a template for my user stories further increases my productivity
- **S4** Using a template for my user stories further increases the quality of my work deliverables
- **S5** Using a quality framework for my user stories further increases my productivity
- **S6** Using a quality framework for my user stories further increases the quality of my work deliverables

Finally, the respondents could optionally provide their contact details and comment on the research and the survey. The survey has been reviewed by two academics who are not part of the authors and was piloted with three practitioners: a developer, a designer and a project manager. Based on the pilot, we revised the survey by adding six questions, removing one question, changing the order of existing questions and making three questions optional.

The goal of the follow-up interviews is to capture the respondent's rationale behind the answers they provided in the survey. The interview protocol consists of 16 questions (see (Lucassen, 2015)). After the preliminaries, the interviewee was asked to explain the role of user stories in their organization and their general perspective on user stories. Next, the respondent was asked to explain the difference between a poor and good user story in his opinion and to clarify their answers to the Likert-type statements **S1**–**S6**.

2.2.2 *Survey Respondents*

Because we posted links to the survey on public venues, it is practically impossible to measure how many individuals we reached. The survey website page garnered 598 unique page views, which Google Analytics defines as *"Unique*

Pageviews is the number of sessions during which the specified page was viewed at least once". These page views led to 197 submitted responses; 6 were duplicates, while others contained impossible or invalid answers such as unclear experience or respondents claiming to be working with user stories since before the year of their introduction. In total, we retained 182 valid responses.

2.2.3 *Follow-Up Interview Respondents*

Out of the 119 respondents (65%) who supplied their email address at the end of the survey, the authors identified 49 respondents that could potentially provide *opinionated* answers in a follow-up interview. We invited all respondents that either (i) provided very positive or very negative answers, (ii) gave varied answers to the Likert-type questions, or (iii) added a comment at the end of the survey. In total, 21 respondents participated, a 43% response rate.

This group of respondents is quite diverse and its composition differs from that of the survey's respondents. Notable differences are that more practitioners participated that work in consultancy (9/21) and/or have the role of requirements engineer/business analyst (6/21). The average interviewee has 6 years of experience with user stories. Respondents originated from 7 different countries; 11 from the Netherlands, 5 from the United States of America, the remaining 5 were all from different countries: Argentina, Brazil, Canada, Portugal and the United Kingdom.

2.3 USER STORY USAGE

This section reports on data collected related to $\mathbf{RQ_1}$ on the use of user stories by practitioners. We examine and report on the first part of the survey results and highlight specific findings from the follow-up interviews. Our twelve key findings are marked within the text as $\mathbf{F_1}$–$\mathbf{F_{12}}$.

2.3.1 *Respondent Context*

As recommended by Cohn (Cohn, 2004), user stories are primarily used in combination with Agile methods. Scrum in particular is used by the majority of respondents. We asked respondents to indicate both which software development methods they used in general, and in which methods they employed user stories. The majority indicate they work with Scrum (94%), but Kanban (40%) and waterfall (29%) are popular as well. XP (13%), V-Model (7%), Spiral (3%) and 14 other methods (9%) are considerably less common. Responses to this

question accentuate the tight coupling of user stories with Agile methods: 99% of respondents that work with Scrum employ user stories - all respondents but two ($\mathbf{F_1}$). As one follow-up interviewee noted: *"For me, user stories and Scrum are interconnected".* Indeed, 17 out of 21 interviewees mention Scrum without it being a subject of discussion. Kanban and XP have a tight coupling as well: 79% and 83% of the respondents that use these software development methods do employ user stories. However, none of the interviewees mention either method during the interview. Users of waterfall and the V-model do not employ user stories often: just 21% and 31% do.

On average, respondents had 4 years of experience with user stories; 57 of them (31%) had more than 5 years of experience. On average, the organizations of the respondents were working with user stories for slightly longer, 4.4 years; 64 (35%) organizations were working with user stories for more than 5 years. Respondent roles include product manager (29%), developer (21%), requirements engineer (18%), software architect/CTO (8%), project manager (8%) and other (16%). Respondents work for fairly uniform organization types: software product (51%), consultancy (20%), custom software (19%) and other (10%). The organization sizes, however, are quite diverse: 1-9 (12%), 10-49 (20%), 50-249 (27%), 250-499 (8%) and 500+ (33%).

Additionally, we asked respondents to self-assess their skill level. The average years of experience per skill level are as follows: Beginner - 1.91 (n=34), Intermediate - 3.05 (n=77), Advanced - 4.76 (n=49), and Expert - 8.95 (n=22). Surprisingly, the aggregate of our respondents did not fall victim to the Dunning-Kruger effect; a cognitive bias which causes individuals with low skill to overestimate their ability and performance in comparison to their highly skilled peers - and vice versa (Kruger and Dunning, 1999).

2.3.2 *The Role of User Stories*

After introductions, the first question of each follow-up interview was to describe the role of user stories in the interviewee's organization. In our 21 interviews, we collected as many different accounts of the role of user stories in their organization. The interviewees explanations range from very close to the approach described Cohn's book (Cohn, 2004) to adaptations that are rather far from agile software development. The majority of interviewees, however, are somewhere in between because they have adapted user story theory to their own situational context. Nevertheless, all approaches have one crucial aspect in common: the user story is the most granular representation of a requirement that developers use to build new features.

2.3.3 *Template*

The use of a template when writing user stories can be considered standard industry practice - only 27 respondents (15%) indicate they do not use a template. The most popular template is the 'original' one (Cohn, 2004). 59% of respondents utilize the Connextra template ($\mathbf{F_2}$): *"As a ⟨role⟩ , I want ⟨goal⟩, [so that ⟨benefit⟩]"*. An additional 10% of respondents use the identical template, but without the *"[so that ⟨benefit⟩]"* clause. The remaining 32 respondents (18%) are spread between 15 approaches, none of which have a significant share. One of these templates omits the role, including only the what and the why.

In the follow-up interview, respondents were asked to explain whether they have a specific reason for using the template they use. Out of the 19 interviewees that use a template just one decided to study and select the most appropriate template for his situation. The remaining interviewees were taught or heard of a specific template at some point and never encountered the need to change to another template. This is likely a factor in explaining the prevalence of the Connextra template.

2.3.4 *Quality Guidelines*

The use of quality guidelines is commonplace among practitioners. The most well-known framework is INVEST (Wake, 2003), which posits that a good user story has the following characteristics: Independent, Negotiable, Valuable, Estimatable, Small and Testable. 33% of respondents indicate they follow self-defined quality guidelines when writing user stories, while 23.5% use the standardized INVEST approach. 39.5% of respondents do not validate their user stories with any form of quality guidelines. The remainder use alternatives, or indicate that it depends on the situation (4%).

When asked to explain what their self-defined quality guidelines entail, all 10 interviewees admit they do not have a well-defined, structured list of concerns they consult when writing user stories. Instead, they rely on the experience of the user story writer and multiple rounds of peer review to ensure the quality of their user stories. Interviewees that do not use quality guidelines, indicate this is not a conscious decision but rather that they are not aware of quality guidelines like INVEST ($\mathbf{F_3}$).

2.4 PERCEPTION OF USER STORY EFFECTIVENESS

This section investigates $\mathbf{RQ_2}$: how practitioners perceive the effectiveness of user stories. We examine the second part of the survey to report on how practitioners perceive the impact of user stories, templates and quality guidelines in terms of their productivity and work deliverable quality.

As expected, the collected data is not normally distributed, making parametric statistics that rely on testing means inappropriate for our Likert-type questions (Clason and Dormody, 1994). Instead, we treat the answers as ordinal data. To report on central tendency and variability of ordinal data, Boone and Boone (Boone and Boone, 2012) recommend using the median or mode and frequencies. To confirm that the variability is from independent populations, Boone and Boone recommend using the statistical χ-square test for independence. Throughout the remainder of this section, applying this test enables us to determine whether a specific variable influences the outcome of the Likert-type questions.

2.4.1 *User Stories in Isolation*

Both the median and mode of the Likert-style questions indicate that practitioners *agree* that representing requirements as user stories and following a template increase their work productivity and deliverable quality. For quality guidelines, both median and mode are *neutral* for gained productivity and quality. For more insights regarding practitioners' opinion on user stories we examine the frequency distributions in Fig. 2.1. In the subsequent subsections,

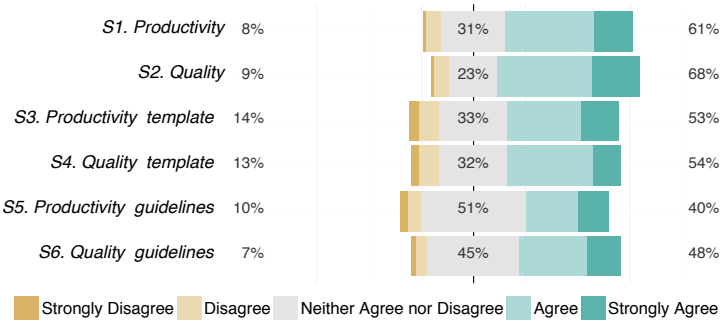


Figure 2.1: Perception of user story effectiveness. The shown percentages (left-to-right) refer to Strongly Disagree + Disagree, Neither Agree nor Disagree and Agree + Strongly Agree. The same format is used in the following charts.

we analyze specific slices of the data using frequency distributions and the χ-square test for independence.

Examining the frequency distribution of our respondents' answers one observation stands out: only a fraction of respondents perceive user stories, templates and quality guidelines to be detrimental to their work productivity and deliverable quality (the percentages on the left of Fig. 2.1 are all between 7% and 14%). Even when we consider neutral answers as negative, the majority of respondents agree or strongly agree that user stories and templates improve work productivity ($\mathbf{S_1}$: 61%, $\mathbf{S_3}$: 53%) and quality ($\mathbf{S_2}$: 68%, $\mathbf{S_4}$: 54%). Respondents are ambivalent about quality guidelines: 51% and 45% indicate they neither agree nor disagree that quality guidelines improve work productivity $\mathbf{S_5}$ or quality $\mathbf{S_6}$. During the follow-up interviews, respondents were asked to clarify their answers. From their comments on user stories in general, we present the following common sentiments to show how the interviewees perceive user stories.

The right software ($\mathbf{F_4}$): 10 interviewees mention that user stories are an enabler for developing the *right* software. In their experience, the technical quality of software does not improve by using user stories and neither do they directly impact the speed of software development. In fact, user stories require more work upfront because the stakeholders have to decompose a requirement into small, comprehensible chunks. This decomposition, however, forces all stakeholders to think and talk about the details of a requirement. This builds a common understanding within the team of what the end-user expects of the software. Thanks to the identification of the *right* requirements, developers are enabled to create the *right* software. According to the literature, this may prevent defects which cost 10-200 times as much to correct later in the software development lifecycle (Boehm, 1988; McConnell, 2001). One interviewee reported that user stories force developers to meet the customer numerous times, resulting in code that is very close to customer expectations. This improves productivity, despite the significant amount of time that is devoted to interacting with the customer.

"User stories optimize for happiness"(Bedell, 2006) ($\mathbf{F_5}$): 5 interviewees do not view productivity or quality gains as essential contributions of user stories. Other aspects of agile development methods have a bigger impact on these concerns. The real advantage of user stories is that stakeholders enjoy working with user stories, fostering a pleasant work environment.

2.4.2 *The Role Of Using a Template*

The first data slice we examine concerns respondents that follow a template for user stories (n=155) versus those who do not (n=27). The frequency distributions in Fig. 2.2 show that respondents using a template more often agree that user stories improve work deliverable quality (71% vs. 52%). However, because the two populations are not independent in a statistically significant manner ($\alpha = .187$), we cannot claim that respondents who use a template are more positive towards user stories.

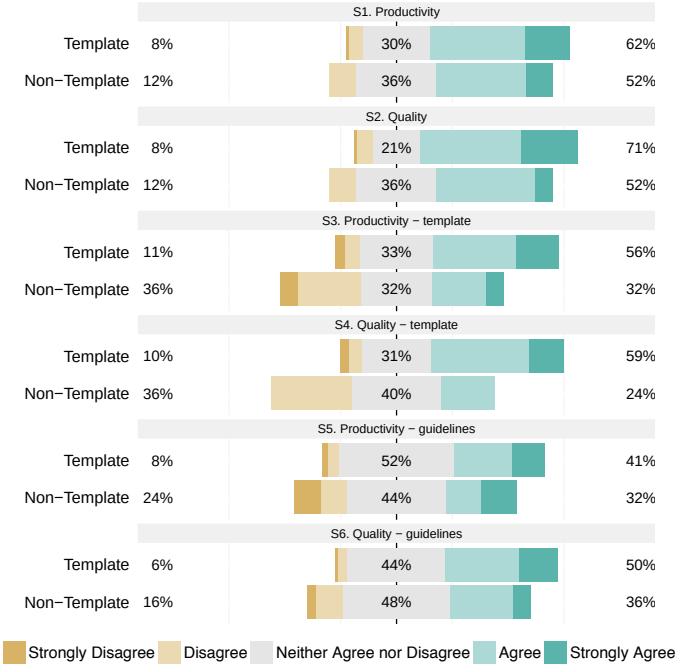


Figure 2.2: Perception of respondents that use a template vs. those that don't.

For the statements on the impact of templates on work productivity ($\mathbf{S_3}$) and quality ($\mathbf{S_4}$), the populations are statistically independent with α's of .02 and .00. Indeed, the difference is striking on both the negative (11% and 9% vs. 33% and 37%) and positive (57% and 60% vs. 30% and 22%) sides of the distribution. These results indicate that respondents that use templates agree considerably more often that using a template contributes

to productivity and work deliverable quality. The question is, however, if this difference is an objective judgment or is rather due to the fact that the respondents are persuaded by the choice of using a template. During the follow-up interviews, we asked respondents to clarify why or how they believe that templates contribute to work productivity and quality. They shared the following comments:

***A* template, not *the* template (F_6):** 12 interviewees mention the beneficial impact of a standard structure for defining user stories. Recall, however, that in Sec. 2.2.3 all interviewees but one did not have an explicit motivation for using the template they use. 3 respondents remark that it does not matter which particular template is used. The use of *a template*, any template is what makes the difference. A single, agreed upon template ensures that everyone within a team works in the same way. When a team can rely upon a standardized structure, their alignment improves overall work productivity and quality. This quote by one respondent effectively illustrates why: *"It's not the template that improves quality, it's what we're doing - we're sharing requirements and a template makes that easy to do and more likely that we'll do it"*.

The why is essential (F_7): While the most popular template for user stories considers the *"[so that ⟨benefit⟩]"* or *why* section as optional, our respondents emphasize the importance of this part for reaping the full rewards of user stories. They attribute a variety of benefits to the inclusion of the purpose of a user story, which lead to work productivity and quality improvements. Adding the *why* part: (1) alleviates confusion among stakeholders, (2) reduces the amount of discussion necessary and (3) provides developers with autonomy in their work. This is, however, easier said than done. The *why* is difficult to find, as the following quote demonstrates: *"Typically, the why question is correctly answered if after the initial answer, you ask 'why?' again for three more times"*.

A developer with a negative opinion of user stories shared that in his experience business people will abuse a template to formulate the same old requirement in a different format. He complained that user stories become *"a blanket way to generally describe what the solution is the company has already defined for you"*, which conflicts with the principle that requirements should be problem-oriented (Lucassen et al., 2015; Zave and Jackson, 1997).

2.4.3 *The Impact of Using Quality Guidelines*

The second data slice looks at the perceptions of respondents that follow self-defined quality guidelines (n=60), INVEST (n=43) or none (n=72)[1]. Examining the frequency distributions in Fig. 2.3, we see that respondents that follow quality guidelines are more positive than those that do not ($\mathbf{F_8}$). The χ-square tests for independence of $\mathbf{S_1}$, $\mathbf{S_4}$, $\mathbf{S_5}$ and $\mathbf{S_6}$ are statistically significant; meaning that we can claim that respondents using quality guidelines more often agree that user stories and quality guidelines improve productivity, and templates and quality guidelines further improve work deliverable quality.

The positive attitude of respondents that apply INVEST is remarkable. During the interviews, these respondents were capable of effectively arguing both for and against any productivity and quality gains. Their ideas can be summarized as follows:

INVEST is not a checklist ($\mathbf{F_9}$): 3 interviewees mention that although the INVEST mnenomic can be used as a checklist, interviewees do not use it as such. Instead, the six characteristics of a good user story are internalized by the team and whenever a user story violates INVEST, a team member brings this up for discussion.

INVEST is useful for inexperienced teams ($\mathbf{F_{10}}$): 2 interviewees indicate they primarily use INVEST as a training tool for inexperienced teams. INVEST's comprehensiveness is an effective starting-point for getting product owners started and the development team to understand how to judge user stories. After two or three months, however, stakeholders have sufficient experience with writing and interpreting user stories that the necessity of INVEST diminishes.

[1]Note that 7 responses are excluded. These respondents gave unique 'other' answers, whose samples are too small for statistical analysis.

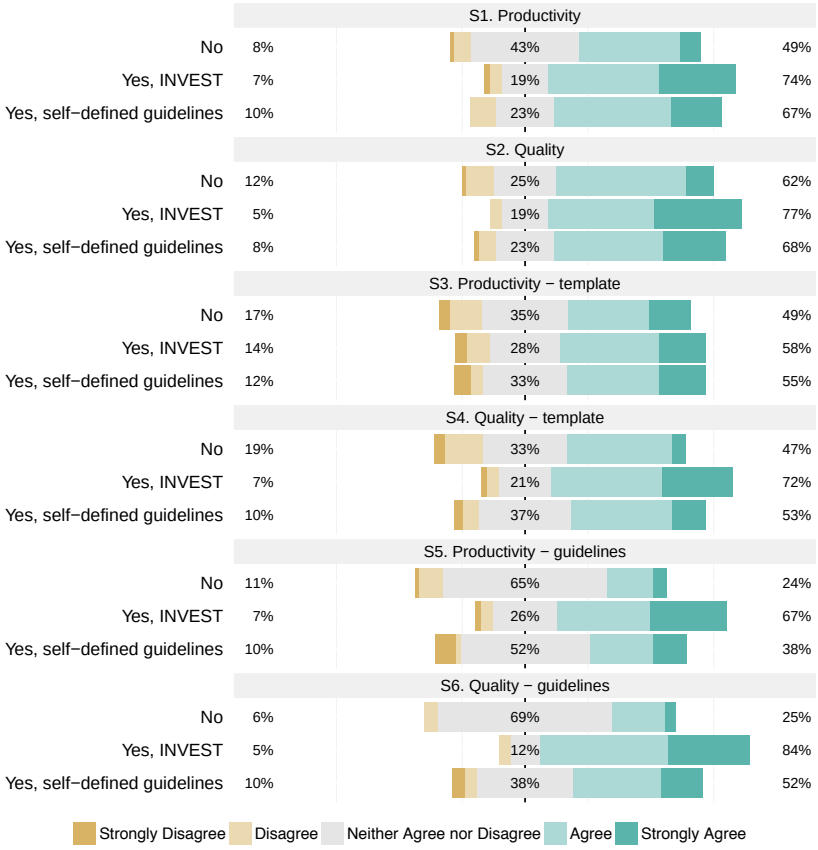


Figure 2.3: Perception of respondents that use INVEST, self-defined quality guidelines or none.

2.4.4 *Technical vs. Non Technical Roles*

To analyze the difference in perception between technical (n=55) and non-technical stakeholders (n=127) we categorize respondents by their role. Because the majority of respondents chose from a pre-defined list of roles, we could easily do this by designating roles containing the term 'software' as technical and those without it as non-technical. The former primarily consists of developers, software architects and CTOs, while the latter includes everything else such as consultants, product managers and the occasional agile coach.

Approximately 60% of both stakeholder types agree with $\mathbf{S_1}$ that user stories improve productivity, while for the other 5 statements non-technical stakeholders are more positive (Fig. 2.4). The average positivity difference between technical and non-technical stakeholders is 22%. For $\mathbf{S_4}$ ($\Delta = 26\%$), $\mathbf{S_5}$ ($\Delta = 28\%$) and $\mathbf{S_6}$ ($\Delta = 25\%$) the populations are independent with statistical significance (α's of .02, .001 and .003) ($\mathbf{F_{11}}$). During follow-up interviews technical respondents were ambivalent about the impact of user stories on their work productivity and quality. In their experience, software development is not necessarily significantly faster nor do they face less bugs.

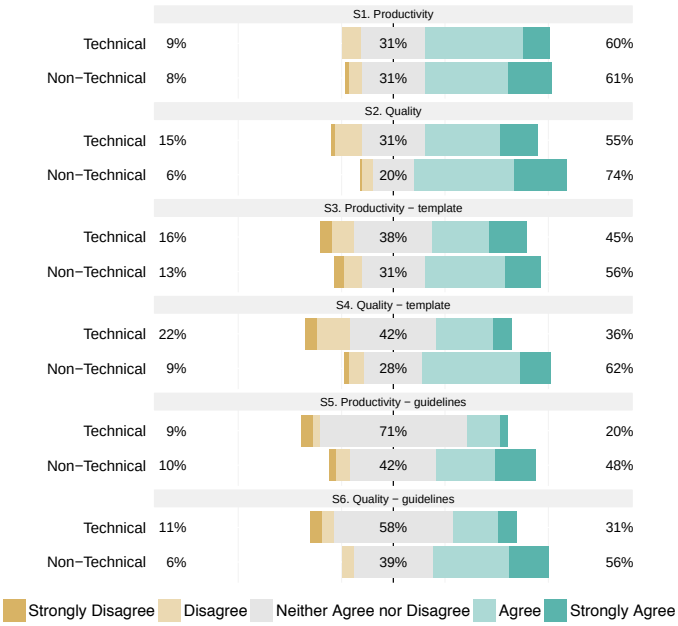


Figure 2.4: Perception of respondents with technical and non technical roles.

2.4.5 *Influence of Expertise Judgement*

For one of the contextual questions we asked respondents to self-assess their user story skill level. They could choose from 5 levels of expertise: novice, beginner, intermediate, advanced and expert. Because only 2 people chose novice, for this analysis we counted them as beginners. Studying the frequency distributions in Fig. 2.5, a pattern catches the eye: as respondents gain more expertise they select *neither agree nor disagree* less frequently, instead opting to agree that work deliverable quality and productivity improves thanks to user stories ($\mathbf{S_1}$ and $\mathbf{S_2}$) and quality guidelines ($\mathbf{S_5}$ and $\mathbf{S_6}$) ($\mathbf{F_{12}}$). This difference is particularly striking when comparing beginners to experts. From a statistical perspective, the answers to $\mathbf{S_1}$ and $\mathbf{S_2}$ on user stories are from independent populations for all four expertise levels. This statistic implies that the difference in their answers cannot be attributed to chance, but that each population has a different perception.

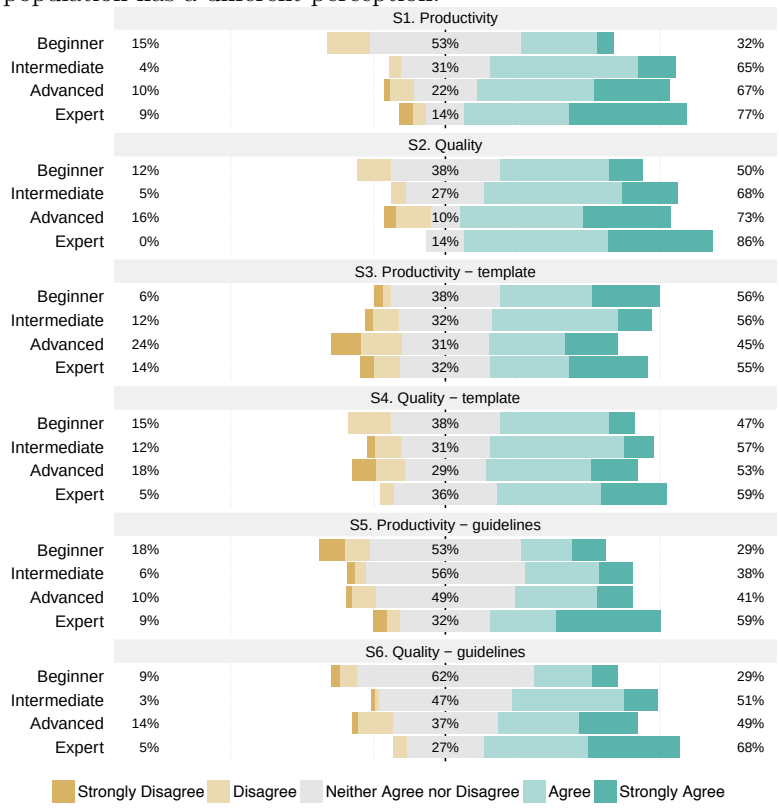


Figure 2.5: Attitude differences per expertise level of respondents.

2.5 VALIDITY THREATS

External validity: Many of the respondents to the survey came from the direct networks of the authors of this paper. Because our research group is focused on the software industry, 93 respondents (51%) are employed by a product software company. Furthermore, 98 respondents (54%) are from the Netherlands. Both have the potential to introduce a bias, which would impact the validity of the results. Examining their frequency distributions(Lucassen, 2015), we see that the percentage differences in the two comparisons are relatively small. Indeed, the χ-square tests for both threats results in significance values between .36 and .78, which is far above the significance threshold of .05. This means that both population pairs are not significantly different and these threats to validity do not hold.

In terms of its composition, the interviewee population is not representative of the survey respondents. In particular, the number of vocally negative interviewees is underrepresented. Although all negative survey respondents were invited for a follow-up interview, there is likely a self-selection process at play. To mitigate this issue, we positively discriminated remarks from negative respondents for inclusion resulting in the abuse paragraph in Sec. 2.4.2.

Internal validity: One of the pre-requisites for participating in the survey was that the respondent expresses requirements as user stories. This decision introduces a selection bias for the respondent population. Potential respondents that decided not to employ user stories or stopped employing user stories are excluded from expressing their views. Thus, our results are generalizable only to user story practitioners.

The follow-up interviews were semi-structured. When an interviewee gave a long answer, the interviewer would summarize the answer and confirm with the interviewee if it was correct. In a small number of cases an experimenter bias occurred, including additional information in the summarization, followed by a potential acquiescence bias - better known as yea-saying. When detected during categorization of the transcriptions, these statements have been ignored.

Construct validity: The survey purposefully did not clearly define what we mean by *productivity* and *work deliverable quality*. Although metrics for quality and productivity in RE exist (e.g., (Lombriser et al., 2016)), these metrics were not appropriate for this survey because of our focus on practitioners' *perception*, and there is no general agreement yet on which specific factors do determine these qualities in RE. Additionally, a key phrase in $\mathbf{S_{3-6}}$ was *further* as in "using a template for my user stories *further* increases my productivity". However, it is impossible to confirm that all respondents fully understood the nuance that they were supposed to evaluate *'using a template'* disjoint from

the user story concept itself. Although a significant threat to validity, we have reason to believe this does not invalidate the results. When the researcher put extra emphasis on this distinction during the follow-up interviews, none of the respondents indicated they misunderstood the question. Nevertheless, we cannot claim that the Likert-type questions are 100% mutually exclusive and exhaustive. Readers should view the survey results as an exploratory evaluation of practitioner's perception of user stories.

The survey contained questions on the subjective terms *expertise*, *quality guidelines*, *role*, *software development method* and *template*. To ensure a uniform interpretation and response, each question was accompanied by standard answers. Because respondents first had to read these, all free-form 'other' answers are expressed in a similar form to the examples. In the case of *quality guidelines*, an additional link to a webpage explaining the INVEST framework was included for additional context. Furthermore, the focus of this study was the *card* aspect of user stories. We purposefully put less emphasis on the *conversation* and *confirmation* as explained by Ron Jeffries (Jeffries), which we will study in greater detail in the future.

2.6 RELATED LITERATURE: USER STORIES, AND PERCEPTION AND EXPERIMENTS IN RE

Between 2003 and 2013, the adoption of user stories has grown tremendously (Kassab, 2015). In agile software development user stories are the predominant method to capture requirements (Wang et al., 2014). Despite their popularity, research efforts concerning user stories are limited. Recent work has revisited user stories from a conceptual perspective. Wautelet et al. propose a unified model for user stories with associated semantics based on a review of 85 user story templates and accompanying example stories (2014). Gomez and colleagues propose a method for identifying dependencies between User Stories (2010). In an earlier paper, we presented a conceptual model that characterizes the structure of a valid user story and decomposes its parts linguistically. This conceptual model is the foundation upon which we built the Quality User Story Framework that proposes quality criteria that a user story should adhere to (Lucassen et al., 2015).

Liskin et al. investigate the expected implementation duration of user story as a characteristic of granularity. They find that in practitioners' experience combining the effort estimation of two small, clear-cut user stories produces more accurate results than when estimating a single, larger, more opaque user story (2014). Multiple authors have linked user stories with goals. Lin et

al. (2014) propose a mixed top-down and bottom-up method where an initial top-down analysis of the high-level goals is complemented by a bottom-up approach that derives more refined goals by analyzing user stories. A similar attempt has been implemented in the US2StarTool (Mesquita et al., 2015), which derives skeletons of i^* goal models starting from user stories. The key difference is that these models represent user stories as social dependencies from the role of the user stories to the system actor.

The number of papers that examine how practitioners use and perceive RE methods and artifacts is limited. Rouibah and Al-Rafee conducted a similar study to ours, investigating the "awareness", "use" and "perceived value generated" of 19 RE techniques of 87 Kuwaiti survey respondees (2009). Their findings include that the most used requirements elicitation techniques are interviews and surveys, but that the highest perceived value comes from decision trees, goal-oriented elicitation and prototyping. Other studies that study perception and use in the context of RE have a different focus. Hofmann and Lehner report on the self-perceived quality of RE service and RE products within RE teams without distinguishing between RE methods (2001). Abrahão et al. present a method to evaluate requirements modeling methods by gauging end-user perceptions, an adaptation of the Method Evaluation Model, and apply it to a Rational Unified Process extension that provides specific techniques for specifying functional requirements (2011).

Nevertheless, the effectiveness of an RE method or technique is a frequent subject of academic literature. In fact, up to four different systematic reviews are available for some RE subdomains. For example, Dieste and Juristo conducted a systematic review on the effectiveness of requirements elicitation techniques and found sufficient evidence to formulate five usage guidelines, such as: unstructured interviews output more complete information than introspective techniques such as protocol analysis (2011). Condori-Ferandez et al. did a systematic mapping study on empirical evaluations of software requirements specification techniques and found that most papers report on experiments that took place in academic environments (2009). The number of experiments conducted with actual practitioners is low. For example, Cruz-Lemus et al. conducted an experiment with practitioners to assess how composite states impact UML statecharts' understandability (2007). Their results are slightly more outspoken with a population of practitioners than a population of students. Penzenstadler, Eckhardt and Fernández even conducted two replication studies to validate their earlier evaluation of an artifact-based RE approach and tool (2013). These studies confirm that their simpler artifact model improves the quality of the created artifacts and ease of use.

2.7 DISCUSSION AND CONCLUSION

This paper has explored how practitioners that already employ user stories use and perceive them. Both the data from our survey with 182 valid responses and comments by follow-up interviewees indicate that software professionals are predominantly positive about *user stories* as well as the associated constructs *templates* and *quality guidelines*. Very few practitioners are downright negative about user stories. Our key findings on user stories are that:

- **F1** Most of the user story adopters (94%) use them in combination with Scrum.
- **F2** The most prevalent user story template is the 'original' one proposed by Connextra.
- **F3** Self-defined quality guidelines are unstructured and not using any quality guidelines is not a conscious decision.
- **F4** The simple structure of user stories enables developing the *right* software, for it facilitates creating a common understanding concerning the requirement.
- **F5** Stakeholders enjoy working with user stories, as they foster a pleasant workplace.
- **F6** Using *a template* benefits RE, not *the template* that the team chooses.
- **F7** Specifying the *why* part of a user story is essential for requirements quality.
- **F8** Practitioners who use the INVEST quality guidelines are significantly more positive about the impact of user stories on productivity and the impact of templates on work deliverable quality.
- **F9** INVEST is not a checklist, but a work guideline each team member should adopt.
- **F10** INVEST is particularly useful for inexperienced teams. The necessity of INVEST diminishes for experienced teams.
- **F11** Technical stakeholders are less positive about the effectiveness of templates and quality guidelines than non-technical stakeholders.
- **F12** Practitioners with more expertise with user stories perceive them more positively.

We discuss **F4**, **F7**, and **F8** in more detail. Throughout the interviews, respondents repeatedly mention that user stories help them create the right software. By requiring all stakeholders to think and talk about the details of a requirement, user stories build a common understanding of what the end-user expects of the software within a team. This identification of the right require-

ments enables development of the right software. This prevents expensive rework, improving productivity and work deliverable quality. Based on this finding, we hypothesize that using user stories reduces software development costs. An associated finding is the importance of the *why* part of a user story to deliver a common understanding and to support development of the right software. This confirms the fundamental theories in RE on the importance of the 'why' for software (process) analysis (Potts and Bruns, 1988; Lee and Lai, 1991; Yu and Mylopoulos, 1994).

There also appears to be a correlation between relying on quality guidelines and the perception of user stories. Respondents that use INVEST are particularly positive in comparison to those that do not apply quality guidelines at all. A clear indication that having a structured list of characteristics of a good user story is beneficial. Recall, however, that our interviewees' self-defined quality guidelines are unstructured, informal approaches and that they are unaware of structured approaches like INVEST. Because of this, we call for an increase in the diffusion of knowledge concerning quality guidelines in order to further improve the positive perception of user stories.

This evaluation of practitioner's use and perception of user stories opens avenues for future research. To test whether adopting user stories reduces software development costs, we are planning to conduct a series of experiments. To improve the diffusion of structured quality guidelines like INVEST or the QUS Framework (Lucassen et al., 2015) we need to conduct a more thorough evaluation of their impact on software development. In particular, studies that take into account the opinion of practitioners that chose not to employ user stories or stopped employing user stories would fill a gap created by this work. Furthermore, despite user stories' increasing popularity, little to no advanced methods and tools originating from academia support them. As adoption of user stories increases, the importance of and opportunities for designing advanced methods and tools for user stories intensifies. We call for academia to focus more resources on user stories and its related concepts.

ACKNOWLEDGEMENTS

The authors would like to thank all survey respondents for participating in our research, the three respondents to the pilot survey as well as Leo Pruijt and Erik Jagroep for reviewing drafts of this paper.

Improving Agile Requirements: The Quality User Story Framework and Tool

3

User stories are a widely adopted requirements notation in agile development. Yet, user stories are too often poorly written in practice and exhibit inherent quality defects. Triggered by this observation, we propose the Quality User Story (QUS) Framework, a set of 13 quality criteria that user story writers should strive to conform to. Based on QUS, we present the Automatic Quality User Story Artisan (AQUSA) software tool. Relying on natural language processing (NLP) techniques, AQUSA detects quality defects and suggest possible remedies. We describe the architecture of AQUSA, its implementation, and we report on an evaluation that analyzes 1,023 user stories obtained from 18 software companies. Our tool does not yet reach the ambitious 100% recall that Dan Berry and colleagues require NLP tools for RE to achieve. However, we obtain promising results and we identify some improvements that will substantially improve recall and precision.

This work was originally published as:

G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Improving Agile Requirements: The Quality User Story Framework and Tool. *Requirements Engineering*, 21 (3):383–403, 2016b

3.1 INTRODUCTION

User stories are a concise notation for expressing requirements that is increasingly employed in agile requirements engineering(Cao and Ramesh, 2008) and in agile development. Indeed, they have become the most commonly used requirements notation in agile projects (Kassab, 2015; Wang et al., 2014), and their adoption has been fostered by their use in numerous books about agile development (Beck, 1999; Jeffries et al., 2000; Beck and Fowler, 2000; Cohn, 2004). Despite some differences, all authors acknowledge the same three basic components of a user story: (1) a short piece of text describing and representing the user story, (2) conversations between stakeholders to exchange perspectives on the user story, and (3) acceptance criteria.

The short piece of text representing the user story captures only the essential elements of a requirement: *who* it is for, *what* is expected from the system, and, optionally, *why* it is important. The most widespread format and de-facto standard (Lucassen et al., 2016a), popularized by Mike Cohn (Cohn, 2004) is: *"As a ⟨type of user⟩ , I want ⟨goal⟩, [so that ⟨some reason⟩]"*. For example: *"As an Administrator, I want to receive an email when a contact form is submitted, so that I can respond to it"*.

Despite this popularity, the number of methods to assess and improve user story *quality* is limited. Existing approaches either employ highly qualitative metrics, such as the six mnemonic heuristics of the INVEST (Independent-Negotiable-Valuable-Estimatable-Scalable-Testable) framework (Wake, 2003), or generic guidelines for quality in agile RE (Heck and Zaidman, 2014). We made a step forward by presenting the Quality User Story (QUS) framework (originally proposed in (Lucassen et al., 2015)), a collection of 13 criteria that determine the quality of user stories in terms of syntax, pragmatics, and semantics.

We build on the QUS framework and present a comprehensive, tool-supported approach to assessing and enhancing user story quality. To achieve this goal, we take advantage of the potential offered by natural language processing (NLP) techniques. However, we take into account the suggestions of Daniel Berry, Ricardo Gacitua, Peter Sawyer and Sri Fatimah Tjong (Berry et al., 2012) on the criticality of achieving 100% recall of quality defects, sacrificing precision if necessary. We call this the *Perfect Recall Condition*. If the analyst is assured that the tool has not missed any defects, (s)he is no longer required to manually recheck the quality of *all* the requirements.

Existing state-of-the-art NLP tools for RE such as QuARS (Bucchiarone et al., 2005), Dowser (Popescu et al., 2008a), Poirot (Cleland-Huang et al.,

2007) and RAI (Gacitua et al., 2010) take the orthogonal approach of maximizing their accuracy. The ambitious objectives of these tools demand a deep understanding of the requirements' contents (Berry et al., 2012). However, this is still practically unachievable unless a radical breakthrough in NLP occurs (Ryan, 1993). Nevertheless, these tools serve as an inspiration and some of their components are employed in our work.

Our previous paper (Lucassen et al., 2015) proposed the QUS framework for improving user story quality and introduced the concept of the Automated Quality User Story Artisan (AQUSA) tool. In this paper, we make three new, main contributions to the literature:

- We revise the Quality User Story (QUS) framework based on the lessons learned from its application to different case studies. QUS consists of 13 criteria that determine the quality of user stories in terms of syntax, semantics, and pragmatics.
- We describe the architecture and implementation of the AQUSA software tool, which uses NLP techniques to detect quality defects. We present AQUSA version 1 that focuses on syntax and pragmatics.
- We report on a large-scale evaluation of AQUSA on 1,023 user stories, obtained from 18 different organizations. The primary goals are to determine AQUSA's capability of fulfilling the Perfect Recall Condition with high-enough precision, but also acts as a *formative evaluation* for us to improve AQUSA.

The remainder of this paper is structured as follows. In Section 3.2, we present the conceptual model of user stories that serves as the baseline for our work. In Section 3.3, we detail the QUS framework for assessing the quality of user stories. In Section 3.4, we describe the architecture of AQUSA and the implementation of its first version. In Section 3.5, we report on the evaluation of AQUSA on 18 case studies. In Section 3.6, we build on the lessons learned from the evaluation and propose improvements for AQUSA. Section 3.7 reviews related work. Section 3.8 presents conclusions and future work.

3.2 A CONCEPTUAL MODEL OF USER STORIES

There are over 80 syntactic variants of user stories (Wautelet et al., 2014). Although originally proposed as unstructured text similar to use cases (Beck, 1999) but restricted in size (Cohn, 2004), nowadays user stories follow a strict, compact template that captures *who* it is for, *what* it expects from the system, and (optionally) *why* it is important (Wautelet et al., 2014).

When used in Scrum, two other artifacts are relevant: *epics* and *themes*. An epic is a large user story that is broken down into smaller, implementable user stories. A theme is a set of user stories grouped according to a given criterion such as *analytics* or *authorization* (Cohn, 2004). For simplicity, and due to their greater popularity, we include only epics in our conceptual model.

Our conceptual model for user stories is shown in Figure 3.1 as a class diagram. A user story itself consists of four parts: one role, one means, zero or more ends and a format. In the following subsections, we elaborate on how to decompose each of these. Note that we deviate from Cohn's terminology as presented in the introduction, using the well known means-end (Simon, 1996) relationship instead of the ad hoc goal-reason. Additionally, observe that this conceptual model includes only aggregation relationships. Arguably a composition relationship is more appropriate for a single user story. When a composite user story is destroyed, so are its role, means and end(s) parts. However, each separate part might continue to exist in another user story in a set of user stories. Because of this difficulty in conceptualizing, we choose to use aggregation relationships because it implies a weaker ontological commitment.

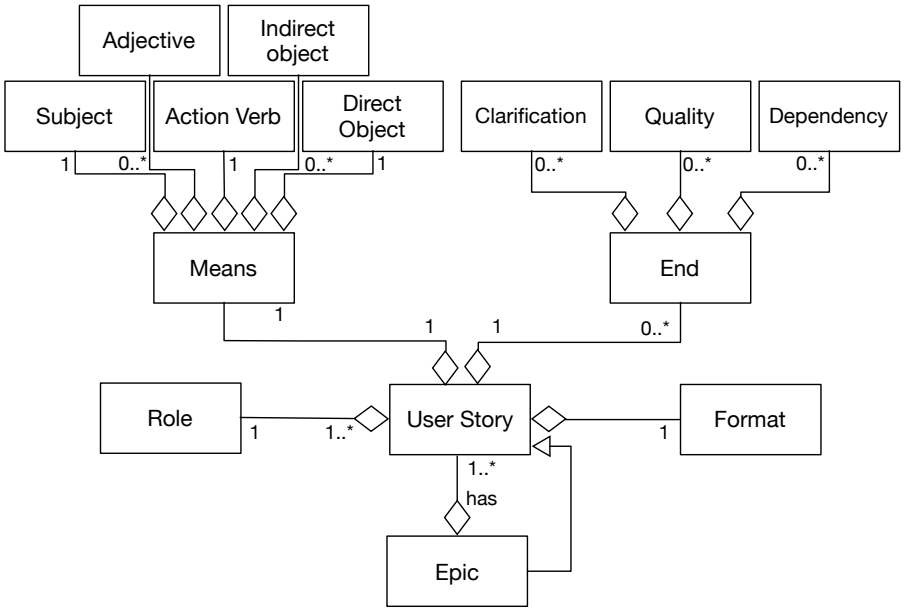


Figure 3.1: Conceptual model of user stories

3.2.1 *Format*

A user story should follow some pre-defined, agreed upon template chosen from the many existing ones (Wautelet et al., 2014). The skeleton of the template is called *format* in the conceptual model; in between which the role, means and optional end(s) are interspersed to form a user story. See the introduction for a concrete example.

3.2.2 *Role*

A user story always includes one relevant role, defining what stakeholder or persona expresses the need. Typically, roles are taken from the software's application domain. Example stakeholders from the ERP domain are Account Manager, Purchaser and Sales Representative. An alternative approach is to use personas, which are named, fictional characters that represent an archetypal group of users (Cooper, 1999). Although imaginary, personas are defined with rigor and precision to clearly capture their goals. Examples are Joe the carpenter, Alice the mother and Seth the young executive, who all have different goals, preferences and constraints. When used in a user story, the name of the persona acts as the role: *"As a Joe"* or *"As an Alice"*.

3.2.3 *Means*

Means can have different structures, for they can be used to represent different types of requirements. From a grammatical standpoint, we support means that have three common elements:

1. a *subject* with an aim such as 'want' or 'am able',
2. an *action verb*[1] that expresses the action related to the feature being requested, and
3. a *direct object* on which the subject executes the action.

For example: *"I want to open the interactive map"*. Aside from this basic requirement, means are essentially free form text which allow for an unbounded number of constructions. Two common additions are an adjective or an indirect object, which is exemplified as follows: "*I want to open a larger* (adjective) *view of the interactive map from the person's profile page* (indirect object)". We included these interesting cases in the conceptual model, but left out all other variations, which we are currently examining in a different research project.

[1]While other types of verbs are in principle admitted, in this paper we focus on action verbs, which are the most used in user stories requesting features

3.2.4 *End*

One or more end parts explain why the means (Cohn, 2004) are requested. However, user stories often also include other types of information. Our analysis of the ends available in the data sets in our previous work (Lucassen et al., 2015) reveals at least three possible variants of a well-formed end:

1. *Clarification of means.* The end explains the reason of the means. Example: *"As a User, I want to edit a record, so that I can correct any mistakes"*.
2. *Dependency on another functionality.* The end (implicitly) references a functionality which is required for the means to be realized. Although dependency is an indicator of a bad quality criteria, having no dependency at all between requirements is practically impossible (Wake, 2003). There is no size limit to this dependency on the (hidden) functionality. Small example: *"As a Visitor, I want to view the homepage, so that I can learn about the project"*. The end implies the homepage also has relevant content, which requires extra input. Larger example: *"As a User, I want to open the interactive map, so that I can see the location of landmarks"*. The end implies the existence of a landmark database, a significant additional functionality to the core requirement.
3. *Quality requirement.* The end communicates the intended qualitative effect of the means. For example: *"As a User, I want to sort the results, so that I can more easily review the results"* indicates that the means contributes maximizing easiness.

Note that these three types of end are not mutually exclusive, but can occur simultaneously such as in *"As a User, I want to open the landmark, so that I can more easily view the landmark's location"*. The means only specifies that the user wishes to view a landmark's page. The end, however, contains elements of all three types: (1) a clarification the user wants to open the landmark to view its location, (2) implicit dependency on landmark functionality, and (3) the quality requirement that it should be easier than other alternatives.

3.3 USER STORY QUALITY

The IEEE Recommended Practice for Software Requirements Specifications defines requirements quality on the basis of eight characteristics (IEEE, 1994): correct, unambiguous, complete, consistent, ranked for importance/stability, verifiable, modifiable and traceable. The standard, however, is generic and it is well known that specifications are hardly able to meet those criteria (Glinz, 2000). With agile requirements in mind, the Agile Requirements Verification Framework (Heck and Zaidman, 2014) defines three high-level verification criteria: completeness, uniformity, and consistency & correctness. The framework proposes specific criteria to be able to apply the quality framework to both feature requests and user stories. Many of these criteria, however, require supplementary, unstructured information that is not captured in the primary user story text.

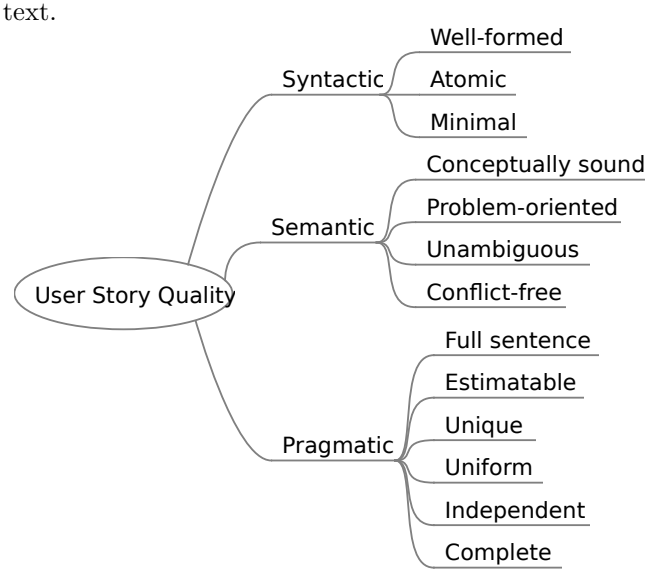


Figure 3.2: Quality User Story Framework that defines 13 criteria for user story quality: overview

With this in mind, we introduce the Quality User Story (QUS) Framework (Figure 3.2 and Table 3.1). The QUS Framework focuses on the intrinsic quality of the user story text. Other approaches complement QUS by focusing on different notions of RE quality such as performance with user stories (Lombriser et al., 2016) or broader requirements management concerns

Table 3.1: Quality User Story Framework that defines 13 criteria for user story quality: details

Criteria	Description	Ind/Set
Syntactic		
Well-formed	A user story includes at least a role and a means	Ind
Atomic	A user story expresses a requirement for exactly one feature	Ind
Minimal	A user story contains nothing more than role, means and ends	Ind
Semantic		
Conceptually sound	The means expresses a feature and the ends expresses a rationale	Ind
Problem -oriented	A user story only specifies the problem, not the solution to it	Ind
Unambiguous	A user story avoids terms or abstractions that lead to multiple interpretations	Ind
Conflict-free	A user story should not be inconsistent with any other user story	Set
Pragmatic		
Full sentence	A user story is a well-formed full sentence	Ind
Estimatable	A story does not denote a coarse-grained requirement that is difficult to plan and prioritize	Ind
Unique	Every user story is unique, duplicates are avoided	Set
Uniform	All user stories in a specification employ the same template	Set
Independent	The user story is self-contained and has no inherent dependencies on other stories	Set
Complete	Implementing a set of user stories creates a feature-complete application, no steps are missing	Set

such as effort estimation and additional information sources such as descriptions or comments (Heck and Zaidman, 2014). Because user stories are a controlled language, the QUS framework's criteria are organized in Lindland's categories (1994):

Syntactic quality, concerning the textual structure of a user story without considering its meaning;

Semantic quality, concerning the relations and meaning of (parts of) the user story text;

Pragmatic quality, considers the audience's subjective interpretation of the user story text aside from syntax and semantics.

The last column of Table 3.1 classifies the criteria depending on whether they relate to an individual user story or to a set of user stories.

In the next subsections, we introduce each criterion by presenting an explanation of the criterion as well as an example user story that violates the specific criterion. We employ examples taken from two real-world user story databases

of software companies in the Netherlands. One contains 98 stories concerning a tailor-made web information system. The other consists of 26 user stories from an advanced healthcare software product for home care professionals. These databases are intentionally excluded from the evaluation of Section 3.5, for we used them extensively during the development of our framework and tool.

3.3.1 *Quality of an individual user story*

We first describe the quality criteria that can be evaluated against an individual user story.

Well-formed

Before it can be considered a user story, the core text of the requirement needs to include a role and the expected functionality: the *means*. US_1 does not adhere to this syntax, as it has no role. It is likely that the user story writer has forgotten to include the role. The story can be fixed by adding the role: *"As a Member, I want to see an error when I cannot see recommendations after I upload an article"*.

Atomic

A user story should concern only one feature. Although common in practice, merging multiple user stories into a larger, generic one diminishes the accuracy of effort estimation (Liskin et al., 2014). The user story US_2 in Table 3.2 consists of two separate requirements: the act of clicking on a location, and the display of associated landmarks. This user story should be split into two:

- US_{2A}: *"As a User, I'm able to click a particular location from the map"*;
- US_{2B}: *"As a User, I'm able to see landmarks associated with the latitude and longitude combination of a particular location."*

Minimal

User stories should contain a role, a means, and (optimally) some ends. Any additional information such as comments, descriptions of the expected behavior or testing hints should be left to additional notes. Consider US_3: aside from a role and means, it includes a reference to an undefined mockup and a note on how to approach the implementation. The requirements engineer should move both to separate user story attributes like the description or comments, and retain only the basic text of the story: *"As a care professional, I want to see the registered hours of this week"*.

Table 3.2: Sample user stories that breach quality criteria from two real-world cases

ID	Description	Violated Qualities
US_1	I want to see an error when I cannot see recommendations after I upload an article	**Well-formed** the role is missing
US_2	As a User, I'm able to click a particular location from the map and thereby perform a search of landmarks associated with that latitude longitude combination	**Atomic**: two stories in one
US_3	As a care professional, I want to see the registered hours of this week (split into products and activities). See: Mockup from Alice NOTE: - First create the overview screen - Then add validations	**Minimal**: there is an additional note about the mockup
US_4	As a User, I want to open the interactive map, so that I can see the location of landmarks	**Conceptually sound**: the end is a reference to another story
US_5	As a care professional I want to save a reimbursement. - Add save button on top right (never grayed out)	**Problem-oriented**: Hints at the solution
US_6	As a User, I am able to edit the content that I added to a person's profile page	**Unambiguous**: what is content?
US_7	As a User, I'm able to edit any landmark	**Conflict-free**: US_7 refers to any landmark, while US_8 only to those that user has added
US_8	As a User, I'm able to delete only the landmarks that I added	
US_9	Server configuration	**Well-formed, full sentence**
US_{10}	As a care professional I want to see my route list for next/future days, so that I can prepare myself (for example I can see at what time I should start traveling)	**Estimatable**: it is unclear what see my route list implies
EP_A	As a Visitor, I'm able to see a list of news items, so that I stay up to date	**Unique**: the same requirement is both in epic EP_A and in story US_{11}
US_{11}	As a Visitor, I'm able to see a list of news items, so that I stay up to date	
US_{12}	As an Administrator, I receive an email notification when a new user is registered	**Uniform**: deviates from the template, no *"wish"* in the means
US_{13}	As an Administrator, I am able to add a new person to the database	**Independent**: viewing relies on first adding a person to the database
US_{14}	As a Visitor, I am able to view a person's profile	

Conceptually Sound

The means and end parts of a user story play a specific role. The means should capture a concrete feature, while the end expresses the rationale for that feature. Consider US_4: the end is actually a dependency on another (hidden) functionality, which is required in order for the means to be realized, implying the existence of a landmark database which is not mentioned in any of the other stories. A significant additional feature that is erroneously represented as an end, but should be a means in a separate user story, for example:

- US_{4A}: *"As a User, I want to open the interactive map"*;
- US_{4B}: *"As a User, I want to see the location of landmarks on the interactive map"*.

Problem-oriented

In line with the problem specification principle for RE proposed by Zave and Jackson (Zave and Jackson, 1997), a user story should specify only the problem. If absolutely necessary, implementation hints can be included as comments or descriptions. Aside from breaking the minimal quality criteria, US_5 includes implementation details (a solution) within the user story text. The story could be rewritten as follows: *"As a care professional, I want to save a reimbursement"*.

Unambiguous

Ambiguity is intrinsic to natural language requirements, but the requirements engineer writing user stories has to avoid it to the extent this is possible. Not only should a user story be internally unambiguous, but it should also be clear in relationship to all other user stories. The Taxonomy of Ambiguity Types (Berry and Kamsties, 2004) is a comprehensive overview of the kinds of ambiguity that can be encountered in a systematic requirements specification. In US_6, *"content"* is a superclass referring to audio, video and textual media uploaded to the profile page as specified in three other, separate user stories in the real-world user story set. The requirements engineer should explicitly mention which media are editable; for example, the story can be modified as follows: *"As a User, I am able to edit video, photo and audio content that I added to a person's profile page"*.

Full Sentence

A user story should read like a full sentence, without typos or grammatical errors. For instance, US_9 is not expressed as a full sentence (in addition to not complying with syntactic quality). By reformulating the feature as a full sentence user story, it will automatically specify what exactly needs to be configured. For example, US_9 can be modified to *"As an Administrator, I want to configure the server's sudo-ers"*.

Estimatable

As user stories grow in size and complexity, it becomes more difficult to accurately estimate the required effort. Therefore, each user story should not become so large that estimating and planning it with reasonable certainty becomes impossible (Wake, 2003). For example, US_{10} requests a route list so that care professionals can prepare themselves. While this might be just an unordered list of places to go to during a workday, it is just as likely that the feature includes ordering the routes algorithmically to minimize distance traveled and/or showing the route on a map. These many functionalities inhibit accurate estimation and call for splitting the user story into multiple user stories; for example,

- US_{10A}: *"As a Care Professional, I want to see my route list for next/-future days, so that I can prepare myself"*;
- US_{10B}: *"As a Manager, I want to upload a route list for care professionals"*.

3.3.2 *Quality of a Set of User Stories*

These quality criteria focus on the quality of a set of user stories; they verify the quality of a complete project specification, rather than analyze an individual story. To make our explanation more precise, we associate every criterion with first-order logic predicates that enable verifying if the criterion is violated.

NOTATION. Lowercase identifiers refer to single elements (e.g., one user story), and uppercase identifiers denote sets (e.g., a set of user stories). A user story μ is a 4-tuple $\mu = \langle r, m, E, f \rangle$ where r is the role, m is the means, $E = \{e_1, e_2, \ldots\}$ is a set of ends, and f is the format. A means m is a 5-tuple $m = \langle s, av, do, io, adj \rangle$ where s is a subject, av is an action verb, do is a direct object, io is an indirect object, and adj is an adjective (io and adj may be null, see Figure 3.1). The set of user stories in a project is denoted by $U = \{\mu_1, \mu_2, \ldots\}$.

Furthermore, we assume that the equality, intersection, etc. operators are semantic and look at the meaning of an entity (e.g., they account for synonyms). To denote that a syntactic operator, we add the subscript *"syn"*; for instance, $=_{syn}$ is syntactic equivalence. The function $depends(av, av')$ denotes that executing the action av on an object requires first executing av' on that very object (e.g., *"delete"* depends on *"create"*).

In the following subsections, let $\mu_1 = \langle r_1, m_1, E_1, f_1 \rangle$ and $\mu_2 = \langle r_2, m_2, E_2, f_2 \rangle$ be two user stories from the set U, where $m_1 = \langle s_1, av_1, do_1, io_1, adj_1 \rangle$ and $m_2 = \langle s_2, av_2, do_2, io_2, adj_2 \rangle$.

Unique and conflict-free

We present these two criteria together because they rely on the same set of predicates that can be used to check whether quality defects exist.

A user story is unique when no other user story in the same project is (semantically) equal or too similar. We focus on similarity that is a potential indicator of duplicate user stories; see, for example US_{11} and epic EP_A in Table 3.2. We can improve this situation by providing more specific stories, for example:

- US_{11A} As a Visitor, I'm able to see breaking news;
- US_{11B} As a Visitor, I'm able to see sports news.

Additionally, a user story should not conflict with any of the other user stories in the database. A requirements conflict occurs when two or more requirements cause an inconsistency (Robinson, 1989; Paja et al., 2013). Story US_7 contradicts the requirement that a user can edit any landmark (US_8), if we assume that editing is a general term that includes deletion too. A possible way to fix this is to change US_7 to: *"As a User, I am able to edit the landmarks that I added"*.

To detect these types of relationships, each user story part needs to be compared with the parts of other user stories, using a combination of similarity measures that are either syntactic (e.g., Levenshtein's distance) or semantic (e.g., employing an ontology to determine synonyms). When similarity exceeds a certain threshold, a human analyst is required to examine the user stories for potential conflict and/or duplication.

Full Duplicate. A user story μ_1 is an exact duplicate of another user story μ_2 when they are identical. This impacts the *unique* quality criterion. Formally,

$$isFullDuplicate(\mu_1, \mu_2) \leftrightarrow \mu_1 =_{syn} \mu_2$$

Semantic Duplicate. A user story μ_1 that duplicates the request of μ_2, while using a different text; this impacts the *unique* quality criterion. Formally,

$$isSemDuplicate(\mu_1, \mu_2) \leftrightarrow \mu_1 = \mu_2 \wedge \mu_1 \neq_{syn} \mu_2$$

Different Means, Same End. Two or more user stories that have the same end, but achieve this using different means. This relationship potentially impacts two quality criteria, as it may indicate: (i) a feature variation that should be explicitly noted in the user story to maintain an *unambiguous* set of user stories, or (ii) a conflict in how to achieve this end, meaning one of the user stories should be dropped to ensure *conflict-free* user stories. Formally, for user stories μ_1 and μ_2:

$$diffMeansSameEnd(\mu_1, \mu_2) \leftrightarrow m_1 \neq m_2 \wedge E_1 \cap E_2 \neq \emptyset$$

Same Means, Different End. Two or more user stories that use the same means to reach different ends. This relationship could affect the qualities of user stories to be *unique* or *independent* of each other. If the ends are not conflicting, they could be combined into a single larger user story; otherwise, they are multiple viewpoints that should be resolved. Formally,

$$sameMeansDiffEnd(\mu_1, \mu_2) \leftrightarrow m_1 = m_2 \wedge (E_1 \setminus E_2 \neq \emptyset \vee E_2 \setminus E_1 \neq \emptyset)$$

Different Role, Same Means and/or Same End. Two or more user stories with different roles, but same means and/or ends indicates a strong relationship. Although this relationship has an impact on the *unique* and *independent* quality criteria, it is considered good practice to have separate user stories for the same functionality for different roles. As such, requirements engineers could choose to ignore this impact. Formally,

$$diffRoleSameStory(\mu_1, \mu_2) \leftrightarrow r_1 \neq r_2 \wedge (m_1 = m_2 \ \vee \ E_1 \cap E_2 \neq \emptyset)$$

End = Means. The end of one user story μ_1 appears as the means of another user story μ_2, thereby expressing both a wish and a reason for another wish. When there is this strong a semantic relationship between two user stories, it is important to add *explicit dependencies* to the user stories, although this breaks the *independent* criterion. Formally, $purposeIsMeans(\mu_1, \mu_2)$ is true if the means m_2 of μ_2 is an end in μ_1:

$$purposeIsMeans(\mu_1, \mu_2) \leftrightarrow E_1 = \{m_2\}$$

Uniform

Uniformity in the context of user stories means that a user story has a format that is consistent with that of the majority of user stories in the same set. To test this, the requirements engineer needs to determine the most frequently occurring format, typically agreed upon with the team. The format f_1 of an individual user story μ_1 is syntactically compared to the most common format f_{std} to determine whether it adheres with the *uniformity* quality criterion. US_{12} in Table 3.2 is an example of a non-uniform user story, which can be rewritten as follows: *"As an Administrator, I want to receive an email notification when a new user is registered"*. Formally, predicate $isNotUniform(\mu_1, f_{std})$ is true if the format of μ_1 deviates from the standard:

$$isNotUniform(\mu_1, f_{std}) \leftrightarrow f_1 \neq_{syn} f_{std}$$

Independent

User stories should not overlap in concept and should be schedulable and implementable in any order (Wake, 2003). For example, US_{14} is dependent on US_{13}, because it is impossible to view a person's profile without first laying the foundation for creating a person. Much like in programming loosely coupled systems, however, it is practically impossible to never breach this quality criterion; our recommendation is then to make the relationship visible through the establishment of an explicit dependency. How to make a dependency explicit is outside of the scope of the QUS Framework. Note that the dependency in US_{13} and US_{14} is one that cannot be resolved. Instead, the requirements engineer could add a note to the backside of their story cards or a hyperlink to their description fields in the issue tracker. Among the many different types of dependency, we present two illustrative cases.

Causality. In some cases, it is necessary that one user story μ_1 is completed before the developer can start on another user story μ_2 (US_{13} and US_{14} in Table 3.2). Formally, the predicate $hasDep(\mu_1, \mu_2)$ holds when μ_1 causally depends on μ_2:

$$hasDep(\mu_1, \mu_2) \leftrightarrow depends(av_1, av_2) \ \wedge \ do_1 = do_2$$

Superclasses. An object of one user story μ_1 can refer to multiple other objects of stories in U, indicating that the object of μ_1 is a parent or *superclass* of the other objects. *"Content"* for example can refer to different types of multimedia and be a superclass, as exemplified in US_6. Formally, predicate $hasIsaDep(\mu_1, \mu_2)$ is true when μ_1 has a direct object superclass dependency based on the sub-class do_2 of do_1.

$$hasIsaDep(\mu_1, \mu_2) \leftrightarrow \exists \mu_2 \in \ U.\ is\text{-}a(do_2, do_1)$$

Complete

Implementing a set of user stories should lead to a feature-complete application. While user stories should not thrive to cover 100% of the application's functionality preemptively, crucial user stories should not be missed, for this may cause a show stopping feature-gap. An example: US_6 requires the existence of another story that talks of the creation of content. This scenario can be generalized to the case of user stories with action verbs that refer to a non-existent direct object: to read, update or delete an item one first needs to create it. We define a conceptual relationship that focuses on dependencies concerning the means' direct object. Note that we do not claim nor believe this relationships to be the only relevant one to ensure completeness. Formally, the predicate $voidDep(\mu_1)$ holds when there is no story μ_2 that satisfies a dependency for μ_1's direct object:

$$voidDep(\mu_1) \leftrightarrow depends(av_1, av_2) \ \wedge \ \nexists \mu_2 \in U.\ do_2 = do_1$$

3.4 THE AUTOMATIC QUALITY USER STORY ARTISAN

The Quality User Story (QUS) Framework provides guidelines for improving the quality of user stories. To support the framework, we propose the Automatic Quality User Story Artisan (AQUSA) tool, which exposes defects and deviations from good user story practice.

In line with Berry et al's notion of a *dumb tool* (Berry et al., 2012), we require AQUSA to detect defects with close to 100% *recall*[2], which is the number of true positives in proportion to the total number of relevant defects. We call this the *Perfect Recall Condition*. When this condition is not fulfilled, the requirements engineer needs to manually check the entire set of user stories for missed defects (Tjong and Berry, 2013), which we want to avoid. On the other hand, *precision*, the number of false positives in proportion to the detected defects, should be high enough so the user perceives AQUSA to report useful errors.

AQUSA is designed as a tool that focuses on easily describable, algorithmically determinable defects: the *clerical* part of RE (Tjong and Berry, 2013). This also implies that the first version of AQUSA focuses on the QUS criteria

[2]Unless mathematically proven, 100% recall is valid until a counterexample is identified. Thus, we decide to relax the objective to "close to 100% recall"

for which the probability of fulfilling the Perfect Recall Condition is high; thus, we include the syntactic criteria and a few pragmatic criteria that can be algorithmically checked, but we exclude semantic criteria as they require deep understanding of requirements' content (Ryan, 1993).

Next, we present AQUSA's architecture, discuss the selected quality criteria including their theoretical and technical implementation in AQUSA v1 as well as example input and output user stories.

3.4.1 *Architecture and Technology*

AQUSA is designed as a simple, stand-alone, deployable as-a-service application that analyzes a set of user stories regardless of its source of origin. AQUSA exposes an API for importing user stories, meaning that AQUSA can easily integrate with any requirements management tool such as Jira, Pivotal Tracker or even MS Excel spreadsheets by developing adequate connectors. By retaining its independence from other tools, AQUSA is capable of easily adapting to future technology changes. Aside from importing user stories, AQUSA consists of five main architectural components (Figure 3.3): Linguistic Parser, User Story Base, Analyzer, Enhancer, and Report Generator.

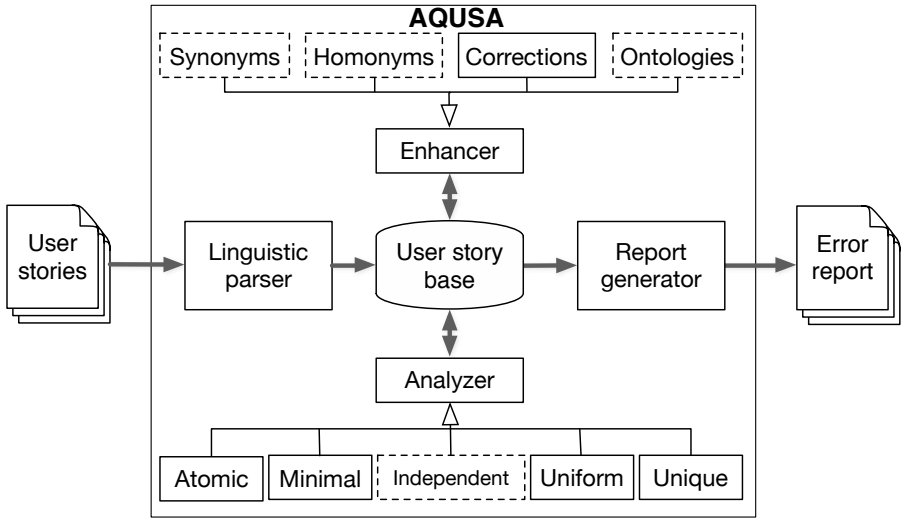


Figure 3.3: Functional view on the architecture of AQUSA. Dashed components are not fully implemented yet

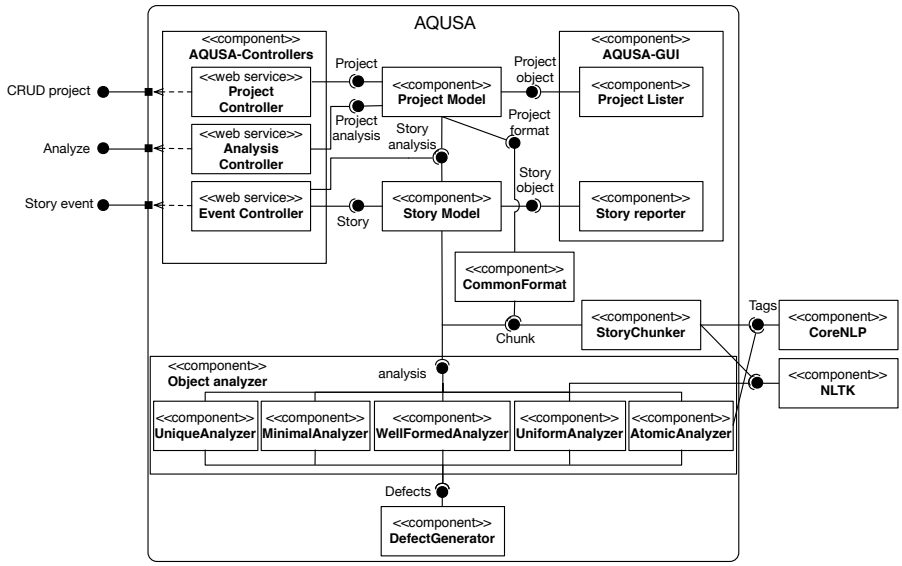


Figure 3.4: Development view on the architecture of AQUSA

The first step for every user story is validating that it is well-formed. This takes place in the linguistic parser, which separates the user story in its role, means and end(s) parts. The user story base captures the parsed user story as an object according to the conceptual model, which acts as central storage. Next, the analyzer runs tailor-made method to verify specific syntatic and pragmatic quality criteria - where possible enhancers enrich the user story base, improving the recall and precision of the analyzers. Finally, AQUSA captures the results in a comprehensive report.

The development view of AQUSA v1 is shown in the component diagram of Figure 3.4. Here we see that AQUSA v1 is built around the model-view-controller design pattern. When an outside requirements management tool sends a request to one of the interfaces, the relevant controller parses the request to figure out what method(s) to call from the Project Model or Story Model. When this is a story analysis, AQUSA v1 runs one or more story analyses by first calling the StoryChunker and then running the Unique-, Minimal-, WellFormed-, Uniform- and Atomic-Analyzer. Whenever one of these encounters a quality criteria violation, it calls the DefectGenerator to record a defect in the database tables associated to the story. Optionally, the

end-user can call the AQUSA-GUI to view a listing of all his projects or a report of all the defects associated with a set of stories.

AQUSA v1 is built on the Flask microframework for Python. It relies on specific parts of both Stanford CoreNLP[3] and the Natural Language ToolKit[4] (NLTK) for the StoryChunker and AtomicAnalyzer. The majority of the functionality, however, is captured in tailor-made methods whose implementation is detailed in the next subsections.

3.4.2 *Linguistic Parser: Well-formed*

One of the essential aspects of verifying whether a string of text is a user story, is splitting it into role, means and end(s). This first step takes place in the linguistic parser (see the functional view) that is implemented by the component *StoryChunker*. First, it detects whether a known, common indicator text for role, means and ends is present in the user story such as 'As a', 'I want to', 'I am able to' and 'so that'. If successful, AQUSA categorizes the words in each chunk by using the Stanford NLP POS Tagger[5]. For each chunk, the linguistic parser validates the following rules:

- Role: Is the last word a noun depicting an actor? Do the words before the noun match a known role format e.g. 'as a'?
- Means: Is the first word 'I'? Can we identify a known means format such as 'want to'? Does the remaining text include at least a second verb and one noun such as 'update event'?
- End: Is an end present? Does it start with a known end format such as 'so that'?

Basically, the linguistic parser validates whether a user story complies with the conceptual model presented in Section 3.2. When the linguistic parser is unable to detect a known means format, it takes the full user story and strips away any role and ends parts. If the remaining text contains both a verb and a noun it is tagged as a 'potential means' and all the other analyzers are run. Additionally, the linguistic parser checks whether the user story contains a comma after the role section. A pseudocode implementation is shown in Algorithm 1. Note that the Chunk method tries to detect the role, means and ends by searching for the provided XXX_FORMATS. When detecting a means fails, it tests whether a potential means is available.

[3] http://nlp.stanford.edu/software/corenlp.shtml
[4] http://www.nltk.org/
[5] http://nlp.stanford.edu/software/tagger.shtml

If the linguistic parser encounters a piece of text that is not a valid user story such as *"Test 1"*, it reports that it is not well-formed because it does not contain a role and the remaining text does not include a verb and a noun. The story *"Add static pages controller to application and define static pages"* is not well-formed because it does not explicitly contain a role. The well-formed user story *"As a Visitor, I want to register at the site, so that I can contribute"*, however, is verified and separated into the following chunks:

Role: As a Visitor
Means: I want to register at the site
End: so that I can contribute

Algorithm 1 Linguistic Parser

```
1: procedure StoryChunker
2:     role = Chunk(raw_text, role, ROLE_FORMATS)
3:     means = Chunk(raw_text, means, MEANS_FORMATS)
4:     ends = Chunk(raw_text, ends, ENDS_FORMATS)
5:     if means==null then
6:         potential_means= raw_text - [role, ends]
7:         if Tag(potential_means).include?('verb' and 'noun')
8:         then means = potential_means
```

3.4.3 *User Story Base and Enhancer*

A linguistically parsed user story is stored as an object with a role, means and ends part—aligned with the first decomposition level in the conceptual model in Figure 3.1—in the user story base, ready to be further processed. But first, AQUSA enhances user stories by adding possible synonyms, homonyms and relevant semantic information—extracted from an ontology—to the relevant words in each chunk. Furthermore, the enhancer has a subpart *corrections* which automatically fixes defects that it is able to correct with 100% precision. For now, this is limited to the good practice of injecting comma's after the role section. AQUSA v1 does not include the other enhancer's subparts.

3.4.4 *Analyzer: Atomic*

To audit that the means of the user story concerns only one feature, AQUSA parses the means for occurrences of the conjunctions "and, &, +, or" in order

to include any double feature requests in its report. Additionally, AQUSA suggests the reader to split the user story into multiple user stories. The user story *"As a User, I'm able to click a particular location from the map and thereby perform a search of landmarks associated with that latitude longitude combination"* would generate a suggestion to be split into two user stories: (1) *"As a User, I want to click a location from the map"* and (2) *"As a User, I want to search landmarks associated with the lat long combination of a location."*.

AQUSA v1 checks for the role and means chunks whether the text contains one of the conjunctions "and, &, +, or". When this is the case, it triggers the linguistic parser to validate that the text on both sides of the conjunction has the building blocks of a valid role or means as defined in Section 3.4.2. Only when this is the case, AQUSA v1 records the text after the conjunction as an atomicity violation.

3.4.5 *Analyzer: Minimal*

To test this quality criterion, AQUSA relies on the results of chunking and verification of the *wellformedness* quality criterion to extract the role and means. When this process has been successfully completed, AQUSA reports any user story that contains additional text after a dot, hyphen, semicolon or other separating punctuation marks. In *"As a care professional I want to see the registered hours of this week (split into products and activities). See: Mockup from Alice NOTE: - First create the overview screen - Then add validations"* all the text after the first dot ('.') AQUSA reports as not minimal. AQUSA also records the text between parentheses as not minimal.

AQUSA v1 runs two separate minimality checks on the entire user story using regular expressions in no particular order. The first searches for occurrences of special punctuation such as "-, ?, ., *". Any text that comes afterwards is recorded as a minimality violation. The second minimality check searches for text that is in between brackets such as "(), [], {},⟨ ⟩" to record as a minimality violation.

3.4.6 *Analyzer: Explicit Dependencies*

Whenever a user story includes an explicit dependency on another user story, it should include a navigable link to the dependency. Because the popular issue trackers Jira and Pivotal Tracker use numbers for dependencies, AQUSA checks for numbers in user stories and checks whether the number is contained within a link. The example *"As a care professional, I want to edit*

the planned task I selected - see 908." would prompt the user to change the isolated number to *"See PID-908"*, where PID stands for the project identifier. In the issue tracker, this should automatically change to *"see PID-908 (http://company.issuetracker.org/PID-908)"*. This explicit dependency analyzer has not been implemented for AQUSA v1. Although it is straightforward to implement for a single issue tracker, we have not done this yet to ensure universal applicability of AQUSA v1.

3.4.7 *Analyzer: Uniform*

Aside from chunking, AQUSA extracts the user story format parts out of each chunk and counts their occurrences throughout the set of user stories. The most commonly occurring format is used as the standard user story format. All other user stories are marked as non-compliant to the standard and included in the error report. For example, AQUSA reports that *"As a User, I am able to delete a landmark"* deviates from the standard 'I want to'.

When the linguistic parser completes its task for all the user stories within a set, AQUSA v1 first determines the most common user story format before running any other analysis. It counts the indicator phrase occurrences and saves the most common one. An overview of the underlying logic is available in Algorithm 2. Later on, the dedicated uniformity analyzer calculates the edit distance between the format of a single user story chunk and the most common format for that chunk. When this number is bigger than 3, AQUSA v1 records the entire story as violating uniformity. We have deliberately chosen 3 so that the difference between 'I am' and 'I'm' does not trigger a uniformity violation, while 'want' vs. 'can' or 'need' or 'able' does.

Algorithm 2 Uniformity Analyzer

```
1: procedure GET_COMMON_FORMAT
2:     format = [ ]
3:     for chunk in ['role', 'means', 'ends'] do
4:         chunks = [ ]
5:         for story in stories do
6:             chunks += extract_indicators(story.chunk)
7:             format += Counter(chunks).most_common(1)
8:     project.format = format
```

3.4.8 *Analyzer: Unique*

AQUSA could implement each of the similarity measures that we outlined in (Lucassen et al., 2015) using the WordNet lexical database (Miller, 1995) to detect semantic similarity. For each verb and object in a means or end, AQUSA runs a WordNet::Similarity calculation with the verbs or objects of all other means or ends. Combining the calculations results in one similarity degree for two user stories. When this metric is bigger than 90%, AQUSA reports the user stories as potential duplicates.

AQUSA v1 implements only the most basic of uniqueness measures: exact duplication. For every single user story, AQUSA v1 checks whether an identical other story is present in the set. When this is the case, AQUSA v1 records both user stories as duplicates. The approach outlined above is part of future work, although it is unlikely to fulfill the Perfect Recall Condition unless a breakthrough in computer understanding of natural language occurs (Ryan, 1993).

3.4.9 *AQUSA-GUI: Report Generator*

The AQUSA-GUI component of AQUSA v1 includes a report generation front-end that enables using AQUSA without implementing a specific connector. Whenever a violation is detected in the linguistic parser or one of the analyzers, a defect is immediately created in the database, recording the type of defect, a highlight of where the defect is within the user story and its severity. AQUSA uses this information to present a comprehensive report to the user. At the top, a dashboard is shown with a quick overview of the user story set's quality showing the total number of issues, broken down into defects and warnings as well as the number of perfect stories. Below the dashboard, all user stories with issues are listed with their respective warnings and errors. See Figure 3.5 on the next page for an example.

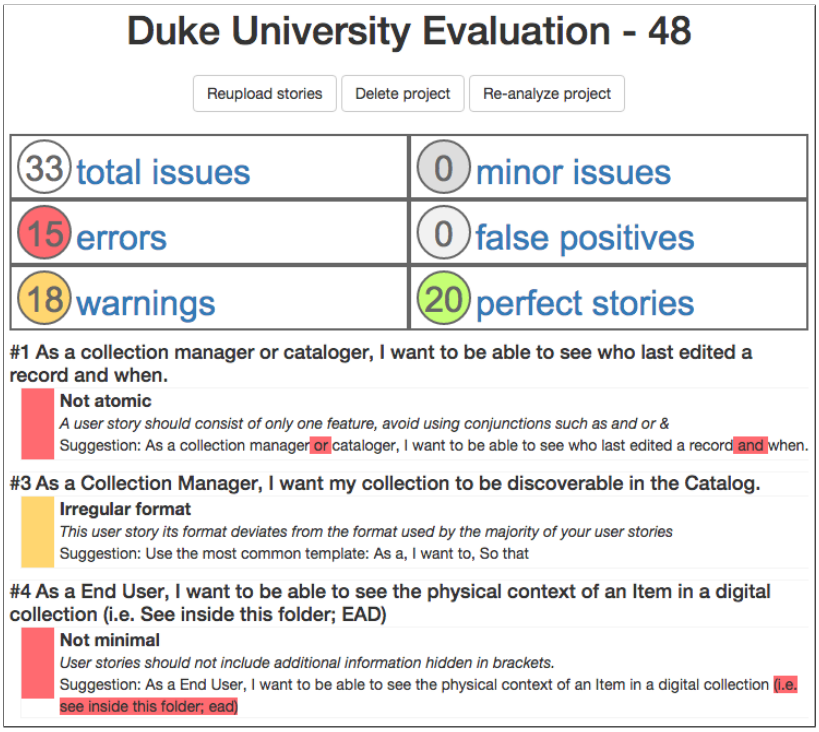


Figure 3.5: Example report of a defect and warning for a story in AQUSA

3.5 AQUSA EVALUATION

We present an evaluation of AQUSA v1 on 18 real-world user story sets. Our evaluation's goals are as follows:

1. To validate to what extent the detected errors actually exist in practice;
2. To test whether AQUSA fulfills the Perfect Recall Condition;
3. To measure AQUSA's precision for the different quality criteria.

The 18 real world user story sets have varying origins. 16 are from medium to large independent software vendors (ISVs) with their headquarters in the Netherlands. 1 ISV is headquartered in Brazil. Although all ISVs create different products focusing on different markets, a number of attributes are in common. For one, all 16 ISVs create and sell their software business-

to-business. In terms of size, 5 ISVs have less than 50 employees, 7 have between 100 and 200 employees and 5 have between 500 and 10,000 employees. Unfortunately, we are unable to share these user story sets and their analyses due to confidentiality concerns. Because of this, we also analyzed a publicly available set of user stories created by a Duke University team for the Trident project[6]. This public dataset and its evaluation results are available online[7]. Note that due to its substantially different origin, this data set has not been incorporated in the overall statistics.

For each user story set a group of two graduate students from Utrecht University evaluated the quality of these user story sets by interpreting AQUSA's reports and applying the QUS Framework. As part of this research project, students investigated how ISVs work with user stories by following the research protocol accompanying the public dataset. Furthermore, the students assessed the quality of the company's user stories by applying the QUS Framework and AQUSA. They manually verified whether the results of AQUSA contained any false positives as well as false negatives and reported these in an exhaustive table as part of a consultancy report for the company. The first author of this paper reviewed a draft of this report to boost the quality of the reports. On top of this, we went even further to ensure the quality and uniformity of the results. An independent research assistant manually rechecked all the user stories in order to clean and correct the tables. He checked the correctness of the reported false positives and negatives by employing a strict protocol:

1. Record a false positive when AQUSA reports a defect, but it is not a defect according to the description in the QUS Framework (Table 3.1).
2. Record a false negative when a user story contains a defect according to the description in the QUS Framework (Table 3.1), but AQUSA misses it.
3. When a user story with a false negative contains another defect, manually fix that defect to verify that AQUSA still does not report the false negative. If it does, remove the false negative. This is relevant in some cases: (1) When a user story is not well-formed, AQUSA does not trigger remaining analyzers; (2) When a minimality error precedes a false negative atomicity error, removing the minimal text changes the structure of the user story which may improve the linguistic parser's accuracy.

[6]http://blogs.library.duke.edu/digital-collections/2009/02/13/on-the-trident-project-part-1-architecture/

[7]http://staff.science.uu.nl/ lucas001/rej_user_story_data.zip

Table 3.3: Results split per data set, showing # of defects (Def), false positives (FP), and false negatives (FN)

	1: ResearchComp			2: ExpenseComp			3: EnterpriseComp			4: DataComp			5: RealProd		
	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN
Atomic	5	2	1	10	4	0	1	1	0	6	0	1	6	3	2
Minimal	6	3	0	3	1	0	25	5	0	4	2	0	16	6	0
Well-formed	6	4	0	1	1	0	33	21	0	2	0	0	0	0	0
Uniform	17	8	0	27	9	0	38	17	0	7	0	0	9	0	1
Unique	2	0	0	0	0	0	0	0	0	0	0	0	2	0	0
SUM	36	17	1	41	15	0	97	45	0	19	2	1	33	9	3
N, prec, recall	50	53%	95%	50	63%	100%	50	55%	100%	23	89%	94%	51	73%	89%

	6: E-ComComp			7: EmailComp			8: ContentComp			9: CMSComp			10: HealthComp		
	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN
Atomic	7	5	0	12	6	0	9	2	3	1	0	1	8	1	2
Minimal	20	6	0	6	0	0	6	0	0	10	0	0	5	1	0
Well-formed	8	8	1	8	0	0	0	0	0	2	0	0	0	0	0
Uniform	33	4	1	36	0	0	34	0	0	35	0	0	11	0	7
Unique	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SUM	68	23	2	62	6	0	49	2	3	48	0	1	24	2	9
N, prec, recall	64	66%	96%	77	90%	100%	50	96%	94%	35	100%	98%	41	92%	71%

	11: AccountComp			12: PharmacyComp			13: SupplyComp			14: IntegrationComp			15: HRComp		
	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN
Atomic	12	2	0	10	3	1	4	2	1	3	2	0	21	7	6
Minimal	0	0	2	1	0	4	2	0	0	1	0	0	52	28	0
Well-formed	0	0	0	0	0	0	0	0	0	0	0	0	44	26	1
Uniform	11	0	0	14	0	9	0	0	0	46	0	0	41	17	0
Unique	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SUM	41	2	2	25	3	14	6	2	1	50	2	0	158	78	7
N, prec, recall	53	95%	95%	47	88%	61%	54	67%	80%	65	96%	100%	207	51%	92%

	16: FilterComp			17: FinanceComp			P1: Duke University								
	# Def	# FP	# FN	# Def	# FP	# FN	# Def	# FP	# FN						
Atomic	42	39	0	13	4	0	10	1	2						
Minimal	5	0	0	25	5	0	4	3	0						
Well-formed	0	0	0	0	0	0	0	0	0						
Uniform	38	0	0	29	0	0	18	0	0						
Unique	2	0	0	6	0	0	0	0	0						
SUM	87	39	0	73	9	0	32	4	2						
N, prec, recall	51	55%	100%	55	88%	100%	48	88%	93%						

3.5.1 *Results*

The quantitative results of this analysis are available in Table 3.3. For each user story dataset, we include:

Def The total number of defects as detected by AQUSA.
FP The number of defects that were in the AQUSA report, but were not actually a true defect.
FN The number of defects that should be in the AQUSA report, but were not.

From this source data we can extract a number of interesting findings. At first glance, the results are promising, indicating high potential for successful further development. The average number of user stories with at least one defect as detected by AQUSA is 56%. The average recall and precision of AQUSA for all the company sets is shown in Table 3.4. Note the differences between the average and weighted average (macro vs. micro) for recall and precision (Tague-Sutcliffe, 1992). This highlights the impact of outliers like #13 SupplyComp, having only 2 violations, 0 false positives and 1 false negative out of 50 user stories. For the micro-average, the number of violations of each set is taken into account, while the macro-average assigns the same weight to every set . This means that #13 SupplyComp its macro-average 67% recall and 100% precision weighs as much as all other results, while for the micro-average calculations its impact is negligible.

Table 3.4: Overall recall and precision of AQUSA v1, computed using both the micro- and the macro average of the data sets

	Recall	Precision
Macro	92.1%	77.4%
Micro	93.8%	72.2%

In total, AQUSA fulfills the desired Perfect Recall Condition for 5 cases, obtains between 90-100% of defects for 6 sets and manages to get between 55% and 89% for the remaining 6. AQUSA's results for precision are not as strong, but this is expected because of our focus on the Perfect Recall Condition. For just 2 sets AQUSA manages to get 100% precision, for 5 sets precision is between 90-100%, 3 sets are only just below this number with 88-90%. In 7 cases, however, precision is rather low with a range of 50-73%. While AQUSA is unable to achieve 100% recall *and* precision for any of the sets, some do come close: for companies 7, 8, 9, 11 and 14, AQUSA v1 achieves recall and precision higher than 90%. We discuss some improvements in Section 3.6.

Table 3.5: Number of defects, false positives, false negatives, recall and precision per quality criterion

n = 1023	**Totals**				
	# Def	# FP	# FN	Rec	Prec
Atomic	170	83	18	82.9%	51.18%
Minimal	187	57	6	95.5%	69.52%
Well-formed	104	60	2	95.7%	42.31%
Uniform	426	55	18	95.4%	87.09%
Unique	30	0	0	100%	100%
SUM	917	255	44	93.8%	72.2%

Looking at the distribution of violations in Table 3.3 and the total number of violations, false positives and false negatives in Table 3.5, a number of things stand out. With the exception of the quality criteria *unique*, the absolute number of false positives lie close to one another. Relatively speaking, however, *well-formed* and *atomic* stand out. Approximately 50-60% of violations as detected by AQUSA are false positives. Similarly, the number of false negatives is particularly large for *atomic*, *minimal* and *uniform*. In the remainder of this section, we investigate the causes for these errors.

Atomic. Throughout the user story sets, the most frequently occurring false positive is caused by the symbol '&' within a role such as: *"As an Product Owner W&O"* and *"As an R&D Manager"* (n=38). As we show in Section 3.6, this can be easily improved upon. The other two main types of false positives, however, are more difficult to resolve: nouns incorrectly tagged as nouns triggering the AtomicAnalyzer (n=18) and multiple conditions with verbs interspersed (n=14).

Tallying the number of false negatives, we find a diversity of causes. The biggest contributor is that forward or backward slashes are not recognized as a conjunction and thus do not trigger the atomic checker (n=5). A more significant issue, however, is that our strategy of checking whether a verb is present on both sides of the conjunction backfired in 2 cases. Specifically, the words 'select' and 'support' were not recognized as a verb by the CoreNLP part-of-speech tagger, which employs a probabilistic maximum entropy algorithm that miscategorized these words as nouns.

Minimal. The primary cause for minimality false positives is the idiosyncratic use of a symbol at the start of a user story such as the asterisk symbol (n=24). Although a fairly easy false positive to prevent from occurring, the fix will introduce false negatives because in some cases a symbol at the start

is an indication of a minimality error. Because our priority is to avoid false negatives, we have to accept these false positives as an unavoidable byproduct of the AQUSA tool. Another frequently occurring error is abbreviations or translations between brackets (n=14). It might be possible to reduce this number with custom methods.

The 7 false negatives for minimality primarily concern idiosyncratic, very specific textual constructs that are unsupported by AQUSA v1. For example, dataset 11 (AccountComp) delivered 2 user stories with superfluous examples preceded by the word 'like'. HealthComp (dataset 10) has 3 very large user stories with many different if clauses and additional roles included in the means and one user story with an unnecessary pre-condition interspersed between the role and means.

Well-formed. The vast majority of false positives is due to unexpected, irregular text at the start of a user story which AQUSA v1 is unable to properly handle (n=32). Examples are: *"[Analytics] As a marketing analyst"* and *"DESIGN the following request: As a Job coach ..."* by company 3 and 15. Although these are not well-formed defects themselves, this text should not be included at all which means the violation itself is not without merit. Nevertheless, AQUSA could improve the way these violations are reported because these issues are also reported as a minimality violation. Similar to the minimality violations, a well-formedness error is also recorded when a symbol such as the asterisk starts the user story (n=24) because AQUSA v1 is unable to detect a role.

There are only 2 false negatives for the well-formedness criterion. Both of these user stories, however, include other defects that AQUSA v1 does report on. Fixing these will automatically remove the well-formedness error as well. Therefore the priority of resolving these false negatives is low.

Uniform. The false positives are caused by a combination of the factors for minimality and well-formedness. Due to the text at the start, the remainder of the user story is incorrectly parsed, triggering a uniformity violation. Instead, these errors should be counted only as a minimal error and the remainder of the story re-analyzed as a regular user story.

The 22 uniformity false negatives are all similar: the user story expresses an ends using an unorthodox format. This can be either a repetition of 'I want to' or a completely unknown like 'this way I'. AQUSA v1 does not recognize these as ends, instead considering them as a valid part of the means - leading to a situation where AQUSA v1 never even tests whether this might be a deviation from the most common format.

UNIQUE. The recall and precision score for all unique measures is 100%. This is because AQUSA v1 focuses only on exact duplicates, disregarding all semantic duplicates. One could argue that the data sets must thus contain a number of false negatives for unique. Unfortunately, we found in our analysis that this is very difficult to detect without intimate knowledge of the application and its business domain. Unsurprising, considering that the importance of domain knowledge for RE is well documented in literature (Zave and Jackson, 1997). Exact duplicates do not occur in the data very often. Only company 11 has 18 violations in its set - the precise reason for why these duplicates are included is unclear.

3.5.2 *Threats To Validity*

We discuss the most relevant threats to validity for our empirical study. For one, there is a *selection bias* in the data. All the analyzed user story sets are supplied by Independent Software Vendors (ISVs). Moreover, the majority of these ISVs originate from and have their headquarters in the Netherlands. This means that the evaluation results presented above might not be generalizable to all other situations and contexts. Indeed, the user stories from a tailor-made software company with origins in a native English speaking country could possess further edge cases which would impact the recall and precision of AQUSA.

Furthermore, the analysis suffers from *experimenter bias* because the quality criteria of the QUS Framework may have different interpretations. Thus, the independent researcher's understanding of the framework impacts the resulting analysis. To mitigate this, the independent researcher received one-on-one training from the first author, immediate feedback after his analysis of the first user story set and was encouraged to ask questions if something was unclear to him. In some cases a subjective decision had to be made, which the independent researcher did without interference from the authors. In general, he would opt for the most critical perspective of AQUSA as possible. Nevertheless, the data set includes some false negatives and false negatives the first author would not count as such himself.

3.6 ENHANCEMENTS: TOWARDS AQUSA V2

To enhance AQUSA and enrich the community's understanding of user stories we carefully examined each false positive and false negative. By analyzing each user story in detail, we identified *seven* edge-cases that can be addressed to achieve a substantial enhancement of AQUSA in terms of precision *and* recall.

3.6.1 *FN: unknown ends indicator*

One of the most problematic type of false negatives is the failure to detect irregular formats because AQUSA is not familiar with a particular ends indicator (instead of the classic '`so that`'). A simple first step is to add the unorthodox formats available in our data set. This tailored approach, however, is unsustainable. We should, thus, make AQUSA v2 customizable, so that different organizations can define their own vocabulary. Moreover, a crowdsourcing feature that invites users to report whenever one of their indicators is not detected by AQUSA should quickly eradicate this problem.

3.6.2 *FN: indicator and chunk repetition*

A particular elusive problem is the repetition of indicators and accompanying role, means or ends chunks. When AQUSA v1 encounters an indicator text, all text afterwards is a part of that chunk until it encounters an indicator text for the subsequent chunk. Consequentially, AQUSA does not raise any red flags when for example (1) a second role chunk is interspersed between the means and ends section like *"As a pharmacist, I want to ..., if ..., if ..., I as a wholesale employee will prevent ..."* or (2) a known means indicator format is used to express an ends as in *"I want to change my profile picture because I want to express myself"*. To solve this problem, AQUSA v2 will scan for all occurrences of indicator texts and generate a warning whenever an indicator type occurs twice in the user story.

3.6.3 *FN: add slash and greater than*

One very simple improvement is to include the forward and backward slash symbols '/' and '\' in the list of conjunctions. In one user story, the greater than symbol '>' was used to denote procedural steps in the user story, prompting us to include this symbol and its opposite '<' in the list of conjunctions as well. Together, these simple improvements reduce the number of atomicity false negatives by one third.

3.6.4 *FN: verb phrases after 'and'*

AQUSA v1 takes a 'smart' approach to detecting atomicity errors. Whenever the atomic analyzer encounters a conjunction like 'and', a POS tagger makes sure a verb is present on both sides of the conjunction. When this is not the case, it is likely that the user story does not include two separate actions. For 3 user stories in our data sets, however, the POS tagger incorrectly tags a verb as a noun introducing a false negative. This is not surprising. Because no available POS tagger is perfect, our approach is guaranteed to not to achieve the Perfect Recall Condition in all cases.

There are two options to resolve this issue. The simple method is to remove this smart approach and simply report all user stories that include a conjunction in the means as violating the atomic quality criteria. The problem is, however, that this introduces a substantial number of false positives. An alternative approach is to include exceptions in the POS tagger for a specific domain. In our dataset we see that the four incorrectly tagged nouns are common actions in software development: select, support, import and export. Compiling such an exception list does not guarantee the prevention of false negatives, but would improve the situation without re-introducing many false positives.

3.6.5 *FP: Symbols and starting text*

Symbols cause the vast majority of false positives in our set of user stories. We want to resolve these without introducing new false negatives. To do this we plan to enhance AQUSA in two ways.

Symbol in role. Many user stories include a reference to a department as part of the role. When AQUSA v2 encounters an ampersand (&) or plus sign (+) in the role chunk, it takes the following approach:

1. Check whether there is a space before or after the ampersand/plus.
2. Count whether the number of characters before and after the ampersand/plus is bigger than a specific threshold such as three.
3. Run the POS tagger to check whether the phrases before and after the ampersand/plus are actual words. Exceptions are 'I' and 'A'.

Only when the answer to two or more of these checks is no, AQUSA v2 records an atomicity violation.

Removing symbol (text). Whenever a user story has text with a symbol before the start of the role chunk, AQUSA v1 is unable to properly apply its

analysis. To resolve this issue, AQUSA v2 will try to remove the symbol and any associated text preceding the first indicator text, check whether a valid user story remains and then rerun its analyses.

3.6.6 *FP: Abbreviations and translations*

The majority of false positives for the minimality quality criterion are caused by an abbreviation or translation of a word in between brackets. To reduce this number, whenever the minimality analyzer detects a single phrase in between brackets it verifies whether the phrase could be an abbreviation of the word or word group immediately before the phrase.

3.6.7 *Expected results*

We expect that introducing these enhancements will generate a substantial improvement in terms of recall and precision. To foresee how substantial this improvement would be, we categorized all false positives and false negatives and removed those that the enhancements should be able to prevent from occurring by conducting a manual analysis of the data set. The results of our analysis are that the new micro-averaged recall and precision for this collection of user story sets would be **97.9%** and **84.8%** (compare this to the values of AQUSA v1: recall 93.8% and precision 72.2%). With these changes, AQUSA would fulfill the Perfect Recall Condition for 9 of the 18 data sets.

3.7 RELATED WORK

We discuss relevant works about the syntax and use of user stories (Section 3.7.1), quality of requirements (Section 3.7.2), and applications of NLP to RE (Section 3.7.3).

3.7.1 *User Stories*

Despite their popularity among practitioners (Wang et al., 2014; Kassab, 2015), research efforts concerning user stories are limited. While scenario-based requirements engineering has been studied since the Nineties (Holbrook, 1990; Loucopoulos and Karakostas, 1995), the earliest research work on user stories proposes their use as the initial artifact in the design of Human-Computer Interaction systems (Imaz and Benyon, 1999), and argues that user stories contain the intention and motives of a user. In later design stages, the authors

propose to transform user stories into the more formal notation of use cases.

The majority of research in the field, however, attempts to create methods and tools that support or improve user story practice. Rees proposes to replace the pen-and-card approach for writing user stories with the DotStories software tool to translate the index card metaphor to a digital environment (Rees, 2002). This tool relies on so called *teglets* which are *"small rectangular areas on a webpage" ... "users can drag teglets around the page in a similar manner to laying out cards on a table"*. Today, most project management tools for agile software development with user stories are built around the same interface design, including Jira, PivotalTracker, Taiga and Trello.

Observing that the simple comparison of a user story with a pair of other user stories is insufficient to accurately estimate user story complexity, Miranda et al. propose a paired comparison estimation approach using incomplete cyclic designs (Miranda et al., 2009). This work reduces the number of necessary comparisons while still producing reliable estimations. In industry, however, planning poker remains the de-facto standard for estimating user story complexity. In a comparative study, Mahnič and Havelja found that the estimates from planning poker played by experts tend to be more accurate than the mean of all individual expert estimates(Mahnič and Hovelja, 2012).

Liskin et al. investigate the expected implementation duration of user story as a characteristic of granularity. They find that in practitioners' experience combining the effort estimation of two small, clear-cut user stories produces more accurate results than when estimating a single, larger, more opaque user story (Liskin et al., 2014). Dimitrijevic et al. qualitatively compare five agile software tools in terms of their functionality, support for basic agile RE concepts and practices, and user satisfaction. They conclude that basic functionality is well supported by tools, but that user role modeling and personas are not supported at all (Dimitrijević et al., 2015).

In line with our conceptual model of Figure 3.1, some authors have linked user stories with goals. Lin et al. (Lin et al., 2014) propose a mixed top-down and bottom-up method where an initial top-down analysis of the high-level goals is complemented by a bottom-up approach that derives more refined goals by analyzing user stories. A similar attempt has been implemented in the US2StarTool (Mesquita et al., 2015), which derives skeletons of *i** goal models starting from user stories. The key difference is that these models represent user stories as social dependencies from the role of the user stories to the system actor.

Other recent work is revisiting user stories from a conceptual perspective. Wautelet et al. propose a unified model for user stories with associated

semantics based on a review of 85 user story templates and accompanying example stories - 20 from academic literature and 65 from practice (Wautelet et al., 2014). Gomez, Rueda and Alarcón propose a conceptual method for identifying dependencies between User Stories, relying on the data entities that stories refer to (Gomez et al., 2010). Although related, these are different approaches from our conceptual model presented in Section 3.2, which does not aim at reconciling and supporting all possible dialects.

3.7.2 *Quality of Requirements*

Multiple frameworks exist for characterizing the quality of (software) requirements. The *IEEE Standard Systems and software engineering – Life cycle processes – Requirements engineering* is the standard body of work on this subject, defining characteristics of a single requirement, sets of requirements as well as linguistic characteristics for individual requirements (IEEE, 2011). Unfortunately, most requirements specifications are unable to adhere to them in practice (Glinz, 2000), although evidence shows a correlation between high-quality requirements and project success (Kamata and Tamai, 2007).

Heck and Zaidman created the *Agile Requirements Verification Framework*, which is a tailor made quality framework for software development in an agile context. Some authors do mention a selection of quality recommendations (Lucia and Qusef, 2010; Dumas-Monette and Trudel, 2014), but the majority of these are generic, organizational pieces of advice for high-level processes. One exception is the rigorous set of validation checks by Zowghi and Gervasi to detect errors in requirements specifications (Zowghi and Gervasi, 2003). As part of their NORPLAN framework, Farid and Mitropoulos propose requirements quality metrics for non-functional requirements (Farid and Mitropoulos, 2013). Aside from ambiguity, however, these metrics are based on the processes related to requirements instead of the actual requirements themselves. Patel and Ramachandran propose the *Story Card Maturity Model*, a CMM-based process improvement model for story cards and their key process areas (Patel and Ramachandran, 2009). They identify maturity levels that consist of 6-7 key process areas with specific activities to obtain that maturity level. An example is the key process area of defining a standard story card structure which defines 7 key attributes to include, or the risk assessment key process area which prescribes the user to (1) understand the problem, (2) assess story card risk and (3) include the risk assessment on the story card. Unfortunately, however, their maturity model has not been validated yet.

3.7.3 *Natural Language Processing for RE*

Applying natural language processing to RE has historically been heralded as the final frontier of RE. Nowadays, the ambitious objective of automation is regarded as unattainable in the foreseeable future (Ryan, 1993; Berry et al., 2012).

Therefore, RE research has applied NLP to specific use cases. Trying to determine and/or improve the quality of requirements documents using NLP is a popular research domain for example. The DODT tool applies NLP and a domain ontology to semi-automatically transform NL requirements into higher quality semi-boilerplate requirements for embedded systems development (Farfeleder et al., 2011). The MaramaAI tool extracts semi-formal *Essential Use Cases* (EUC) models from natural language requirements and enables an end-user to detect inconsistency, incompleteness and incorrectness by visually highlighting the differences to a baseline EUC (Kamalrudin et al., 2011). The EuRailCheck project uses rule-based techniques and ontologies to validate formalized requirement fragments and pinpoint flaws that are not easy to detect in an informal setting (Cimatti et al., 2013). The HEJF tool bases its detection of *"requirements smells"* that are checked against the ISO 29148 (IEEE, 2011) requirements quality standard. Their proposed lightweight approach uses POS-tagging, morphological analysis and dictionaries to detect linguistic errors in requirements. In an evaluation with industry experts, they obtained positive feedback despite the inability to fulfill the Perfect Recall Condition (Femmer et al., 2014).

Some tools look at very specific aspects of parsing natural language requirements. The SREE tool aims to detect a scoped set of ambiguity issues with 100% recall and close to 100% precision by using a lexical analyzer instead of a syntactic analyzer (Tjong and Berry, 2013). Although their precision is only 66%, they argue that using their tool is still faster and more reliable than manually searching for all instances of ambiguity. Yang et al. combines lexical and syntactical analyzers with an advanced technique from the machine learning domain called Conditional Random Fields (CRFs) to detect uncertainty in natural language requirements. They apply their tool to 11 full-text requirements documents and find that it performs reasonably well in identifying uncertainty cues with F-scores of 62% for auxiliaries, verbs, nouns and conjunctions. On the other hand, it under-performs in identifying the scope of detected uncertainty causing the overall F-score to drop to 52% (Yang et al., 2012).

Another class of NLP for RE tools extract specific elements from natural language requirements. The NFR locator (Slankas and Williams, 2013) uses a vector machine algorithm to extract non-functional requirements from install manuals, requests for proposals and requirements specifications. Their approach was twice as effective as a multinomial naïve Bayes classifier. The glossary tool suite by Arora et al. applies a collection of linguistic techniques to automatically extract relevant domain specific concepts to generate a glossary of terms. Their approach outperforms earlier tools in candidate term identification thanks to the application of tailor-made heuristics (Arora et al., 2014). Finally, the Text2Policy tool attempts to extract Access Control Policies (ACP) from natural language documents to reduce the manual effort for this tedious but important security task. Using both syntactic and semantic methods, this tool achieves accuracies ranging between 80-90% for ACP sentence, rule and action extraction (Xiao et al., 2012). The generation of models from natural language requirements has also been studied; for example, Friedrich et al. combined and augmented several NLP tools to generate BPMN models, resulting in an accuracy of 77% on a data set of text-model pairs from industry and textbooks (Friedrich et al., 2011).

Berry et al. argue that these tools suffer from lack of adoption because of their inaccuracy. If the tool provides less than 100% recall, the analyst still has to repeat the entire task manually without any tool support. He proposes—and we support his position—that tools that want to harness NLP should focus on the *clerical* part of RE that software can perform with 100% recall and high precision, leaving thinking-required work to human requirements engineers (Berry et al., 2012).

3.8 CONCLUSION AND FUTURE RESEARCH

In this paper, we presented a holistic approach for ensuring the quality of agile requirements expressed as user stories. Our approach consists of (i) the QUS framework, which is a collection of 13 criteria that one can apply to a set of user stories to assess the quality of individual stories and of the set, and (ii) the AQUSA software tool, that employs state-of-the-art NLP techniques to automatically detect violations of a selection of the quality criteria in the QUS framework.

In addition to laying foundations for quality in agile requirements, the implementation and evaluation of AQUSA on over 1,000 user stories from 18 organizations provides evidence about the viability of the Perfect Recall Condition. According to this condition, NLP tools for RE should focus on the

clerical activities that can be automated with 100% recall and high-enough precision. Our results show that for some syntactic quality criteria of the QUS framework it is possible to achieve results that are close to the Perfect Recall Condition.

Based on our case studies, we have identified a number of easy-to-implement improvements that will be included in AQUSA v2. Although these improvements originate from 18 different cases, we will have to determine whether these changes lead to over-fitting for the datasets that we have studied so far, and if the actual benefit is as good as we expect.

Further research directions exist that future work should address. The effectiveness of the QUS framework as a quality framework for user stories should be studied in case studies and action research, which may lead to further improvements. Longitudinal studies should be conducted to determine the effectiveness of the AQUSA tool while the requirements database is being populated, as opposed to the current case studies where an existing requirements database was imported. To do this, an approach that explains how to embed AQUSA and the QUS Framework in a standard agile development environment is necessary. The challenge of reducing the number of false positives while staying (close to) 100% recall will be a central direction to follow for AQUSA development. To determine whether our approximation of the Perfect Recall condition is sufficient, we will evaluate AQUSA's performance in comparison to human analysts. After all, humans are unable to achieve the Perfect Recall Condition themselves (Berry et al., 2012). Finally, it is necessary to study whether and to what extent the pragmatic and semantic quality criteria can be included in AQUSA, at least to assist the engineering in specific sub-problems for which our recall/precision goals can be met.

ACKNOWLEDGEMENTS

This paper would not have been made possible without the contributions of many people. We would like to thank Floris Vlasveld, Erik Jagroep, Jozua Velle and Frieda Naaijer for providing the initial real-world user story sets. We also thank the graduate students who participated in the course Software Product Management for their hard work in finding user stories to examine. Special thanks go to Marcel Robeer who did great work as independent research assistant. Petra Heck and Andy Zaidman's collection of agile requirements quality literature was of great help for writing Section 3.7.2. Finally, we thank the companies who generously contributed sets of user stories, time and feedback.

Improving user story practice with the Grimm Method: A multiple case study in the software industry

4

[**Context and motivation**] Previous research shows that a considerable amount of real-world user stories contain easily preventable syntactic defects that violate desired qualities of good requirements. However, we still do not know the effect of user stories' intrinsic quality on practitioners' work.
[**Question/Problem**] We study the effects of introducing the Grimm Method's Quality User Story framework and the AQUSA tool on the productivity and work deliverable quality of 30 practitioners from 3 companies over a period of 2 months.
[**Principal ideas/results**] Our multiple case study delivered mixed findings. Despite an improvement in the intrinsic user story quality, practitioners did not perceive such a change. They explained, however, there was more constructive user story conversation in the post-treatment period leading to less unnecessary rework. Conversely, project management metrics did not result in statistically significant changes in the number of comments, issues, defects, velocity, and rework.
[**Contribution**] Introducing our treatment has a mildly positive effect but a larger scale investigation is crucial to decisively assess the impact on work practice. Also, our case study protocol serves as an example for evaluating RE research in practice.

This work was originally published as:

G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Improving user story practice with the grimm method: A multiple case study in the software industry. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 235–252, 2017b

4.1 INTRODUCTION

Thanks to the rapid adoption of agile development practices, approximately 50% of software practitioners capture requirements using the semi-structured natural language (NL) notation of *user stories* (Wang et al., 2014; Kassab, 2015; Lucassen et al., 2016a): "As a ⟨*role*⟩, I want ⟨*goal*⟩, so that ⟨*benefit*⟩".

Despite the simplicity and wide-spread adoption of this requirements engineering (RE) method, practitioners make many mistakes when creating user stories. In previous work, we analyzed 1,000+ real-world user stories and found that 56% contain easily preventable syntactic defects (Lucassen et al., 2016b). Examples include the use of a non-standard template and the specification of multiple features in one user story.

Our solution for practitioners to create high-quality user stories was to introduce two artifacts: the Quality User Story (QUS) framework and the computational linguistics tool Automatic Quality User Story Artisan (AQUSA) (Lucassen et al., 2016b). Although we have empirically confirmed that numerous real-world user stories violate QUS characteristics (Lucassen et al., 2016b), we still lack evidence of the impact of QUS and AQUSA on practitioners' work.

This paper studies the impact of QUS and AQUSA on team communication frequency, team communication effectiveness, work deliverable quality, and work productivity. We do this via a multiple case study where we compare the pre-treatment period with the experimental period (2 months each) with 30 practitioners from 3 companies.

The participants first attended a 2 hour training, received a summary of the training's content and applied their new skills in a training workshop. After we integrated AQUSA with the company's issue tracker, the participants applied QUS and AQUSA in their work activities for two months. During this period, we collected software development process metrics, practitioner perception of the impact, and inherent user story quality. To determine the effect of our treatment, we compared the results against data taken from the issue tracker in the preceding two months. The results of our empirical study are summarized by the following findings:

- The intrinsic user story quality generally increased after the treatment, resulting in fewer violations of the user story qualities prescribed by the QUS framework;
- The perception on quality by practitioners shows a marginal improvement but without reaching statistical significance;
- Despite our accurate measurements of the impact on work practices, we could not identify any significant change, also due to changes in the organizational context.

The next section of this paper outlines the research method. We then present the results in Section 4.3 and review related literature in Section 4.4. Finally, Section 4.5 presents a discussion of the findings and concludes this paper with an outline of future work.

4.2 RESEARCH METHOD

We study the impact of introducing a tool-supported quality framework for user stories in agile development. We do this through a multiple case study with three software product organizations. Over a span of two months, these organizations incorporated two artifacts into their user story practices: QUS and AQUSA.

QUS (Lucassen et al., 2016b) defines the quality of user stories via three types of criteria: (i) *syntactic* (e.g., atomic and well-formed), concerning the textual structure of a user story; (ii) *semantic* (e.g., problem-oriented and unambiguous), which focus on the meaning of user stories; and (iii) *pragmatic* (e.g., unique and uniform), referring to how user stories are being used in RE. AQUSA automatically verifies some QUS criteria using computational linguistics algorithms. In particular, AQUSA supports those criteria for which we can reach close to 100% recall: unique, minimal, well-formed, uniform and atomic. We combine QUS and AQUSA into a package called *Grimm Method*, which provides an easy-to-remember term for practitioners and explains our artifacts' use during user story creation, poker planning, and software development.

We investigate the impact of the Grimm Method (our independent variable) by presenting data that test the following four hypotheses (each defining one dependent variable): *"Applying the QUS framework accompanied by the AQUSA tool" . . .*:

H1 - *"increases communication frequency"*
H2 - *"fosters more effective communication"*
H3 - *"improves work deliverable quality"*
H4 - *"increases work productivity"*

4.2.1 *Grimm Method Treatment Design*

To minimize threats to the internal validity of our study and to ensure uniform data collection, we devised a stepwise treatment design. All the physical materials described in each of the steps below are available online (Lucassen, 2016). Also, video/audio-recordings of the training sessions can be made available upon request.

1. **Baseline measurement -** Two months before applying the treatment we start collecting software development process metrics. We refer to this time frame as the pre-treatment period.
2. **Intake survey -** On the treatment application day, each experiment participant fills out a brief intake survey to learn more about his/her knowledge of user stories and professional experience.
3. **Training -** The experimental treatment consists of a 2-hour training session during which the participants: (1) attend a presentation by the first author on the Grimm Method (Lucassen, 2016), (2) discuss analysis of their stories by AQUSA, (3) apply the QUS framework in a training workshop and (4) receive a summary of the training as reference material.
4. **Experimental period -** Upon completing the training, the team kicks off the two-month experimental period by integrating AQUSA with their issue tracker. From this moment, the team applies QUS in their daily work activities, supported by AQUSA that automatically analyzes any updated or newly added user stories and collects data on company's software development processes.
5. **Exit survey -** At the end of the experimental period each team member completes an exit survey on the impact of introducing the Grimm Method. The exit survey consists of 24 questions to capture the participant's perception of H1–H4, subdivided in four distinct parts. The first and second part include questions on the pre-treatment period and experimental period in isolation, the third part has questions that directly compare the two periods, and the fourth part includes four open questions for respondents to provide further feedback.
6. **Project Manager Evaluation -** Together with the team's project manager we evaluate the experimental period. The goal of this conversation is to validate our interpretation of the data. In particular, we ask the project manager to clarify outliers and to explain context-specific responses to the exit survey.
7. **Interviews -** We go through the responses to the exit survey to identify participants who give inconsistent, particularly positive, remarkably negative and/or opinionated answers. We invite these participants to clarify their answers in a follow-up interview to gather in-depth qualitative data.

These steps result in three types of data to test the hypotheses: (1) the intrinsic user story quality as reported by AQUSA, (2) practitioners' perception of the Grimm Method's impact on work processes and user story quality, and (3) metrics about the software development process. We detail each data type in the following subsections. We employ methodological triangulation to

reduce bias and to strengthen the validity of our results to form a more detailed and balanced picture of the actual situation (Runeson and Höst, 2009).

4.2.2 *Measures*

User Story Quality. The first type of collected data is the intrinsic user story quality reported by AQUSA. As a direct measure of the impact of introducing the Grimm Method, the results indicate whether the experiment participants actually started creating syntactically better user stories. We apply AQUSA analyses to user stories created during the pre-treatment period and experimental period to detect the total number of violations for five quality criteria: well-formed, atomic, minimal, uniform and unique (Lucassen et al., 2016b). Note that this is a subset of the full QUS framework based on the characteristics that can be automatically checked with high precision and recall accuracy. We expect the total number of violations to decrease after introducing the Grimm Method.

Perceived Impact on Work Practice. We collect the second type of data by means of the exit survey and follow-up interviews. In both cases, the experiment participants self-report how they perceive the impact of the Grimm Method on their work. In total, the exit survey includes 12 Likert-Type statements on whether the respondents perceive the user stories to contribute to H1, H2, H3 and H4. Examples include "The user stories improved my productivity" and "User story conversation occurs more frequently since the Grimm Method training". Furthermore, the survey contains 4 questions and statements on respondents' perception of the intrinsic quality of the user stories themselves such as "How would you rate the quality of the user stories?" and "The quality of our user stories is better since the Grimm Method training". For the complete list of survey questions we refer the reader to the online materials (Lucassen, 2016). Note that similar to our previous work (Lucassen et al., 2016a), the survey relies on the participants' own understanding of *productivity* and *work deliverable quality* rather than enforcing our own definition.

Process Metrics. Over the years, a large corpus of software development metrics has been proposed in the literature. The available metrics span from inherent code quality metrics such as *cyclomatic complexity* (McCabe, 1976) to team well-being metrics like the *Happiness Index* (Kniberg, Henrik, 2010). For RE alone, a literature review by Holm et al. (Holm et al., 2015) distilled 298 unique dependent variables from 78 articles. In RE research, however, it is unclear when and why which specific metrics are applicable (Holm et al., 2015).

Although the quality of user stories can potentially impact code quality metrics, the primary intention of user stories is to streamline the processes facilitating software development. By capturing and communicating the discussion around the features to be implemented, user stories aim to enable developers to produce software that stakeholders actually want (Cohn, 2004) and practitioners perceive a key quality of user stories to be enabling creation of the *right software* (Lucassen et al., 2016a). For this reason this study looks at a specific type of metrics related to the (human) *processes* around software development (Madeyski and Jureczko, 2015).

We select the following five metrics from literature for their relevance to our hypotheses and their availability in project management software like Jira[1]:

Formal communication - We measure the frequency of communication in a team (H1) as the number of comments added to the stories completed in a 2-week sprint.

Rework - We measure the amount of rework in a project by calculating the *recidivism rate*: the rate at which user stories move backward in the software development process (Davis, 2015). When a developer assigns the status 'done' to a user story, but it is not complete or has bugs, it is reassigned 'to-do' or 'in progress', causing the recidivism rate to go up. A high recidivism rates is an indicator of ineffective communication causing misunderstanding among the stakeholders. We calculate recidivism rate as: $200 * (Backward/(Forward + Backward))$. Note that we include a multiplier of 200 to get a natural 0-100% range, instead of 0-50 as in (Davis, 2015). This metric measures communication effectiveness thereby relating to H2.

Pre-release defects - The number of bugs added to the issue tracker during a 2-week sprint. Inspired by defect prediction literature (Schröter et al., 2006; Zimmermann et al., 2007; Shihab et al., 2010), which base their metrics on all bugs in a six-month period before a new release. Unlike these works, we focus on the narrow 2-week period of a sprint as Scrum prescribes every sprint to be a potentially *releasable* increment. This metric measures work deliverable quality, which is used to test H3.

Post-release defects - The number of bugs related to user stories in a sprint, as reported in the two sprints after that sprint. This choice is inspired by defect prediction literature, which counts all defects in the first six months after a release. Again, we substantially shorten this time-frame because of Scrum's quick feedback cycle. Moreover, summing the defects as reported in twelve sprints would make the defect count differences

[1] https://www.atlassian.com/software/jira

between consecutive sprints negligibly small. This metric measures work deliverable quality that is used to assess whether H3 holds.

Team productivity - To measure the productivity of the development team (H4) we sum the story points a team completes in a sprint: the so-called *velocity* (Cohn, 2004).

4.2.3 *Experiment Participants*

We announced the experiment within our professional networks. Based on organizational details and selection criteria such as development sprint length and compatibility of the issue tracker with AQUSA, we invited 11 companies to participate and sent them the exact details of the experiment. 5 companies with headquarters in the Netherlands registered for participation. Unforeseen technical difficulties integrating AQUSA with firewalled enterprise editions of Jira reduced the number of companies to 3.

eCommerce Company is a large company with over 2000 employees. One team working on a next generation version of the platform consisting of 6 developers, 2 UX designers and 3 project managers participated.

Health Company is a medium-sized software producing organization that has multiple products for delivering digital healthcare. A team working on a new product consisting of 4 software developers, 1 software architect, 1 UX designer, 1 scrum master and 1 project manager participated.

RealEstate Company is a medium-sized software producing organization that develops a product for a housing cooperative. The participating team of 11 employees has 6 software developers, 3 testers, one product owner and one product manager.

In total, 30 practitioners from six different countries participated in the experiment. The roles of these participants are: 16 software developers, 7 managers, 3 testers, 3 UX designers and 1 CTO. All the teams use Scrum as their primary software development method and employ user stories. All practitioners report they capture user stories in the Connextra template *"As a ⟨role⟩ , I want ⟨goal⟩, [so that ⟨benefit⟩]"*, 11 of which ensure their quality by applying the INVEST framework, while 8 defined their own guidelines instead and the remaining 11 participants do not use any guidelines. Note that the majority of practitioners from RealEstate company indicated they did not use quality guidelines, while 2 selected INVEST and 2 others chose self-defined guidelines. For the eCommerce company, it was a tie between INVEST and self-defined guidelines with 4 each, followed by 3 saying they are not aware of

any guidelines. The Health company participants are more in agreement with 6 employing INVEST and just 1 each choosing self-defined or no guidelines at all. The average participant has 3.1 years of experience in working with user stories while his or her organization has 4.7 years of experience. Concerning their expertise with user stories, 2 participants indicate they are at the novice stage, 5 are beginner, 15 are intermediate and 8 are advanced.

4.3 RESULTS

This section presents the results of our multiple case study. We first look at the change in intrinsic quality between the pre-treatment period and the experimental period (Section 4.3.1). We then investigate the participants' perception of the effectiveness of the Grimm Method (Section 4.3.2). Finally, we analyze the software development process itself by comparing the metric computed on the basis of issue trackers data (Section 4.3.3).

4.3.1 *Intrinsic User Story Quality*

We collected all user stories created by the companies during the pre-treatment period and the experimental period. We analyzed them running AQUSA's defect detection algorithms. We observe a reduced number of defects in the second period in Table 4.1.

Table 4.1: Intrinsic user story quality analysis by AQUSA, comparing the pre-treatment period (*pre*) and the experimental period (*exp*).

	eCommerce		Health		RealEstate	
	Pre	Exp	Pre	Exp	Pre	Exp
Number of user stories	105	71	33	33	103	135
Defects breakdown						
Well Formed: each story has at least a role and a means	63	27	7	4	33	22
Atomic: each story expresses a requirement for *one* feature	10	1	6	2	6	8
Minimal: each story contains only role, means, and ends	20	11	0	0	30	16
Uniform: all written stories employ the same template	38	19	7	13	24	18
Unique: each story is unique, duplicates are avoided	20	5	2	4	0	0
Total defects	151	63	22	23	93	64
Defects per user story	1.44	0.89	0.67	0.70	0.90	0.47

In particular, there are 38.3% fewer defects per user story (see the last row) for eCommerce company and 47.5% fewer defects for RealEstate company in the experimental period. The number of defects remained practically stable for Health company (one additional defect: 23 instead of 22) (4.5% more). Aggregating all the results, the post-period reveals 116 fewer defects for nearly as many user stories. The companies produced 241 user stories in the pre-treatment period with 266 defects versus 239 user stories with 150 defects in the experimental period. Due to the small number of defects for Health company, there is a substantial difference between the 27% macro-average and 43.14% weighted micro-average defect reduction (Tague-Sutcliffe, 1992).

Looking beyond the averages, the defect distribution mostly remained the same for eCommerce and RealEstate companies, while they changed for Health company. The number of uniformity and uniqueness defects (almost) doubled whereas the number of well-formed and atomic defects (more than) halved.

The results suggest that applying the Grimm Method treatment generally had a positive effect on the intrinsic quality of user stories. On the other hand, the improvement is to be expected considering we measure quality with the exact same criteria as prescribed by the treatment. To further test the quality of the textual requirements, we considered using algorithms that assess single sentence complexity. Unfortunately, this field is still immature (Vajjala and Meurers, 2016); while many metrics for text readability exist, they require passages of text with 100+ words. Nevertheless, we applied multiple readability metrics to our user story collections but found no substantial change between the pre-treatment period and the experimental period. For example: the *Gunning fog index*es (Gunning, 1968) of our three sets are 11.0 v 10.8, 12 v 12.1, and 9.3 v 8.7, while the *New Dale-Chall* scores (Chall and Dale, 1995) are 5 v 5.3, 5.3 v 5.4, and 5.5 v 4.9 for eCommerce, Health and RealEstate, respectively.

4.3.2 *Participant Perception*

At the end of the experimental period each participant completed an exit survey on the perceived impact of introducing the Grimm Method. 27 participants submitted a complete and valid survey. Due to the limited sample size we mostly report on the aggregate answers of all respondents; the final paragraph presents a brief per-company discussion. We first analyze the participants' responses, illustrated with quotes from 6 follow-up interviews. After perceived user story quality, we focus on the treatment's impact (H1–H4). Note that we use H#a to highlight questions that directly compare the pre-treatment and experimental periods, and H#b for questions on an individual period.

User story quality. The exit survey asks the participants to rate user story quality on a scale from 0-10 twice: first for the pre-treatment period and then again for the experimental period. Overall, the responses show a small user story quality improvement after applying the treatment: an average score of 6.96 vs. 7.15 with standard deviations of 1.34 vs. 1.29. In total, 9 respondents report a positive user story quality change after the experimental period, 12 report no change at all and 6 indicate the quality of the user stories decreased. However, when analyzing the distribution of responses via a Wilcoxon signed-rank test, we obtain no statistically significant difference between the two periods ($Z = -.984, p = .325$). Thus, concerning the introduction of the Grimm Method's QUS and AQUSA, we could not observe statistically significant changes in practitioner's perception of user story quality.

Yet, when asked to directly compare the pre-treatment period with the experimental period the responses are less neutral. The majority of respondents (16 or 59%) state they agree with the Likert-Type statement "The quality of our user stories is better since the Grimm Method training", while 10 respondents neither agree nor disagree (37%) and just 1 respondent disagrees (4%). Also, 14 respondents (52%) agree with the statement "My satisfaction of our user stories is higher since the Grimm Method training", 11 neither agree nor disagree (41%) and 2 respondents disagree (7%).

In follow-up interviews, we asked respondents to motivate their answers and to explain why they did not report a change in the user story quality score, yet they do agree that the quality of their user stories improved. Two eCommerce Company and two RealEstate Company interviewees emphasize that their reported increase in user story quality was moderate, if present at all. They indicate that introducing the Grimm Method did not result in a meaningful, lasting process change, but that they themselves and their colleagues did become more aware of the relevance of capturing user stories in a diligent manner. The UX designer from eCommerce Company notes *"I believe that someone coming from outside the organization to tell us how you are supposed to do this has had the most influence. It aligns everyone's ideas on user stories. It helps that the method presents everything in a piecemeal fashion."*

These results reveal a discrepancy between the *intrinsic* and *perceived* change in user story quality. While the number of defects dropped by 43.14%, just 9 participants reported a (marginal) positive change in user story quality. Although respondents did respond positively to the Likert-Type statements, follow-up interviews clarified that the respondents do not believe user story quality improved substantially.

One possible explanation for this mismatch is AQUSA's focus on highlighting simple, easy to detect issues. This results in highly accurate yet seemingly trivial output. The importance of fixing a uniformity mistake, for example, can be difficult to comprehend, and those improvements may be regarded as too small to lead to an improvement.

Communication Frequency and Communication Effectiveness. We study participants' perception of conversation by breaking it down into two dimensions: frequency and constructiveness. Respondents do not report a significant change in H1b: communication frequency between the pre-treatment period and experimental period (see Fig. 4.2). 21 respondents did not change their answer after the training (78%), 4 reported a decrease in conversation frequency (15%) and 2 reported an increase (7%). The majority of respondents still experienced excessive communication around user stories; however, 3 out of the 4 participants from eCommerce Company that reported a communication decrease did no longer perceive the amount of conversation to be *excessive*.

The number of respondents that agree with statement H2b "The user stories contributed to constructive conversation concerning the software to be made" after the treatment grew from 14 to 18 (52% to 67% as per Fig. 4.2). Notably, the respondent that indicated he strongly disagreed with the statement for the pre-treatment period agreed with the statement after the treatment. When directly comparing the pre-treatment period with the experimental period, most respondents agreed that conversation was more frequent (H1a, 13 or 48%) *and* was more effective (H2a, 14 or 52%, see Fig. 4.1).

Again, the data exhibits a discrepancy. Participants self-report small communication differences between the pre-treatment period and experimental period, yet agree with statements that communication frequency and effectiveness improved after attending the Grimm Method training. Follow-up interviewees gave diverse motivations of their answers and clarifications of this discrepancy. The answers regarding whether conversation frequency increased or decreased after applying the treatment varied in particular. One respondent indicated the amount of conversation increased by up to 40% while another thought the amount of conversation decreased substantially.

Regardless of the increase or decrease, however, the interviewees agreed on the positive impact on conversation effectiveness. The same eCommerce Company UX designer noted a positive change in communication effectiveness and frequency: *"Previously we lost a lot of time talking about trivial things. By trying to create better user stories as a team the discussion became more focusedm resulting in more in depth conversation on why and how to approach a problem. Although this means more conversation it also saved time"*.

A software developer from eCommerce Company who contributed to the exhibited discrepancy explained his choice as follows: *"I think the conversation effectiveness did change positively, but the Grimm Method training made me more critical of what to expect from user story conversation in terms of effectiveness"*. This explanation highlights an unexpected phenomenon: although the treatment achieved its intended effect, it simultaneously influenced the way we measure that effect. These consequences cancel each other out, leading to a nullification of the impact measurement.

Work Deliverable Quality and Work Productivity. Although respondents reported little change in work deliverable quality (H3b) after the experimental treatment in Fig. 4.1, 10 respondents (37%) agreed with H3a "The quality of my work deliverables is better since the Grimm Method training" in Fig. 4.2. In follow-up interviews, we asked respondents to clarify this discrepancy. Interviewees' responses were unanimous: the technical quality of the developed software did not improve, whereas the treatment clarified the goals of user stories thereby making it easier to develop software with less rework. This sentiment is illustrated by a software engineer from RealEstate Company: *"More effective and efficient conversation on user stories has reduced the amount of surprises later on. As a developer you have the responsibility to continue posing questions until you know what you are working on. The actual work deliverable quality is the same."*

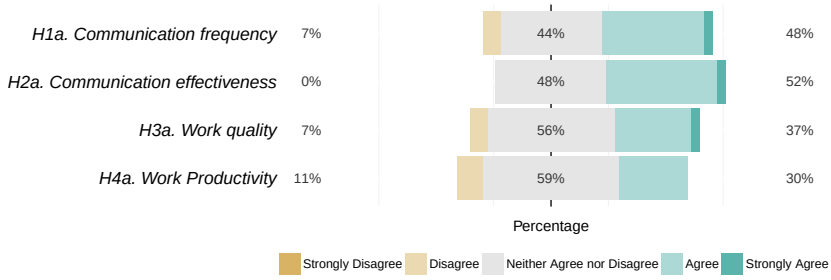


Figure 4.1: Answers to Likert-Type statements directly comparing pre-treatment and experimental periods. The shown percentages (left-to-right) refer to Strongly Disagree + Disagree, Neither Agree nor Disagree and Agree + Strongly Agree, respectively.

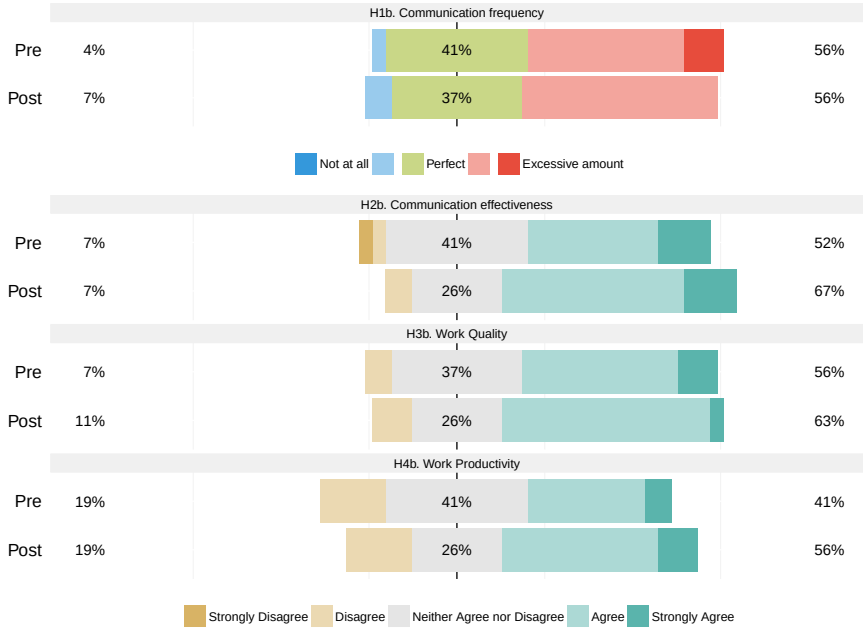


Figure 4.2: Answers to Likert-Type statements for both pre-treatment and experimental periods.

Responses were more consistent for work productivity. For H4b, 7 respondents indicated a positive change (26%), 16 did not change their answer (59%) and 4 experienced a decrease in work productivity (15%). Similarly, 8 or 30% of respondents agreed with H4a "My work productivity is higher since the Grimm Method training" (Fig. 4.2). Again, interviewees did not find user stories to have a direct impact on developers productivity because programming tasks do not substantially change due to better user stories. The interviewees, however, perceived more efficient processes around software development. The product manager of Health Company says *"I do not think the productivity itself increased, but we are more efficient in getting to the stage of being productive"* and a software developer from eCommerce Company *"I do not think my personal productivity increased, but for the entire team it did. As a team you can collaborate and communicate more efficiently and effectively which improves productivity"*.

Note that we found no correlation between a participant's role and his or hers perceived change in work deliverable quality and work productivity. It is likely, however, that this is due to the study's sample size. A future study at a larger scale could confirm our earlier results (Lucassen et al., 2016a) that project managers perceive a more positive impact than software developers. Similarly, there is no statistically significant difference between the three case companies concerning their change in perception due to the limited sample size. Inspecting the Likert-Type graphs per company available online (Lucassen, 2016) reveals that:

- **RealEstate** Company employees clearly indicated a positive impact in all questions concerning both H#a and H#b.
- **Health** Company employees are mostly neutral concerning H#a: slightly positive on communication frequency (H1a) and communication effectiveness (H2a), and mildly negative on work quality (H3a) and work productivity (H4a). In H#b questions, they report no change concerning communication effectiveness (H2b) and work quality (H3b), and a minor positive change about work productivity (H4b).
- **eCommerce** Company employees are in between. They report both positive and negative change after introducing the Grimm Method, with more positive results concerning communication (H1a, H2a, H2b), slightly negative results on work quality (H3b), and neutral results for the other questions.

4.3.3 *Software process metrics*

The raw metric data is presented in Tables 4.2, 4.3 and 4.4, one for each participating company. The columns capture the following data: sprint identifier, number of comments, issues and defects within the sprint, number of post-release defects, recidivism rate and velocity. Note that we separate each table in two: the pre-treatment period corresponds to sprints 1-4 and the experimental period spans across sprints 5-8. A visual, qualitative inspection of the results reveals three main concerns:

1. The total number of comments, issues, defects and velocity varies greatly between companies; compare, for example, the number of comments in the eCommerce company and in the Health company.
2. Some sprints include data outliers such as the number of defects in RE7.
3. The impact of the Grimm Method is likely to be small and can hardly be detected by visual inspection.

Sprint	*Issues*	*Comments*	*Def*	*Post Def*	*Recidivism*	*Velocity*
EC1	50	1	15	24	0.00	24
EC2	67	1	12	18	4.11	34
EC3	59	2	11	10	4.62	8
EC4	71	3	8	6	2.21	15
Treatment applied						
EC5	21	6	3	7	8.00	18
EC6	21	4	2	22	0.00	18
EC7	28	14	4	21	4.84	16
EC8	48	9	17	6	3.06	16

Table 4.2: eCommerce Metric Data

Sprint	*Issues*	*Comments*	*Def*	*Post Def*	*Recidivism*	*Velocity*
HE1	56	54	2	7	14.32	47
HE2	42	64	4	4	9.36	63
HE3	17	26	4	2	3.23	23
HE4	19	46	1	5	7.96	33
Treatment applied						
HE5	20	24	1	12	3.69	32
HE6	21	56	3	12	0.93	50
HE7	30	36	9	3	2.56	37
HE8	26	40	1	0	9.55	42

Table 4.3: Health Metric Data

Sprint	*Issues*	*Comments*	*Def*	*Post Def*	*Recidivism*	*Velocity*
RE1	130	5	9	15	3.57	18
RE2	136	7	1	15	4.05	14
RE3	137	10	4	15	8.23	7
RE4	117	5	2	44	6.67	8
Treatment applied						
RE5	113	12	8	106	3.31	22
RE6	96	2	18	94	7.14	18
RE7	268	18	59	85	8.13	33
RE8	230	9	20	27	6.77	22

Table 4.4: RealEstate Metric Data

To learn the actual impact of the Grimm Method we investigate whether the pre-treatment period and experimental period produce statistically different means. To reduce the numbers' variety we normalize them to a 0 to 100 scale, where 100 is the highest score per columns per company. For example: eCommerce Company's sprint EC7 has the most comments, 14, and is assigned 100. EC6 has 4 comments and gets normalized to $4/14 * 100 = 28.57$.

To test if the mean ranks differ between the pre-treatment and the experimental period we apply the Wilcoxon signed-rank test for non-parametric data (see Table 4.5). None of the tests produce a significant difference between the pre-treatment period and experimental period groups with all $Z < \pm 1.960$. This indicates that introducing the Grimm Method and Tool did not produce a statistically significant change in number of comments, issues, pre-release defects, post-release defects, recidivism rate nor velocity.

	Issues	*Comments*	*Def*	*Post Def*	*Recidivism*	*Velocity*
Z	-1.177	-1.490	-.178	-1.557	-.275	-.628
Sig.	.239	.136	.859	.120	.784	.530

Table 4.5: Wilcoxon signed-rank tests

Although no statistically significant effect is observable, outliers in the data suggest that some change did occur. The outliers' significant divergence from the means make them eligible for removal in case the data has been unfairly influenced. For instance: the number of pre-release defects in EC7 is 3 times as high as the second highest value. We asked the in-charge project managers to clarify the outliers in their data. Their explanations reassured us of their validity. Two of the three project managers attribute the outliers to company-specific irregularities. In summary:

- The eCommerce company team lost productivity starting from EC3 due to internal discussions concerning the product direction.
- For the RealEstate company the merger of two project teams into one in RE3 resulted in productivity loss. Furthermore, an upcoming release in RE7 resulted in the reporting of many defects.
- The Health company project manager did notice a steady increase in productivity during the experimental period. However, he did not believe the Grimm Method has been the primary driver of this improvement, which he attributed to how the team started achieving what it is capable of after a period of under performance.

Yet, each project manager agreed that the frequency and effectiveness of user story conversation improved after the Grimm Method training. A replication of this study on a larger scale is necessary to confirm whether the phenomenon of experiencing a positive change without actively attributing it to the treatment is universal.

4.4 RELATED LITERATURE

Although many evaluations exist on the impact of new RE methods, we could not identify any study where an RE treatment was experimented by comparing months-long periods with companies. A review of empirical papers on software process improvement (Unterkalmsteiner et al., 2012) shows that just 8 out of 148 studies applied experimentation as their research method. None of these 8 both (1) relate to RE and (2) apply a pre-post comparison. However, the literature includes several closely related case study reports.

Kamata and Tamai investigated the relationship between requirements quality and project success or failure (Kamata and Tamai, 2007). They analyzed 32 projects completed in a Japanese company which produces thorough quality reports for all requirements. They detected a weak relationship between SRS quality and project outcome, with five SRS criteria having a strong impact: overview, product perspective, apportioning of requirement, functions and purpose. Similarly, Knauss et al. found that in 40 student projects', success relates to the quality of the SRS they produce. For their specific context, they were even able to define a quality threshold that can be used to predict risk of failure (Knauss et al., 2009).

Damian et al. introduced a formal RE process to the daily work processes of 31 project members of one Australian company (Damian et al., 2005). They collected data over 6 months via a questionnaire, interviews and document inspection to measure practitioner's perception and development performance in terms of estimated effort vs. expended effort. Their analysis of the data indicated a positive effect of improving requirements management process in industry on downstream software development, especially in terms of more accurate estimations, improved project planning, and enhanced project scoping.

Sommerville and Ransom investigated whether improvements in RE process maturity lead to business improvements (Sommerville and Ransom, 2005). Over a period of 18 months, 9 case study companies incorporated advice on RE process improvement and self-reported on business key performance indicators (KPIs). After the experimental period, for each company both the RE process maturity and relevant business KPIs had improved. In spite of this, the authors could not statistically correlate the two due to incompatible KPIs.

Napier et al. explored the feasibility of an RE improvement process based on the RE Good Practice Guide that considers stakeholders' perception of which problems are most relevant (Napier et al., 2009). The authors evaluate this method during a three-year action research process with one company. The results show a 69% increase in the number of implemented RE guidelines and unanimous positive perception by the participants.

Méndez Fernández and Wagner (Méndez Fernández and Wagner, 2015) explored the effect of the RE improvement approach *ArtREPI* that applies the RE best practice database AMDiRE. The impact of ArtREPI in two case studies is measured via two post-treatment questionnaires: one on the support of process engineers and one about project participants' rating of ArtREPI's output. ArtREPI meets practitioners' process improvement needs when problem and artifact orientation are important, but the authors call for larger-scale replications.

Our research incorporates elements from each of the aforementioned studies. We consider the relevance of little direct author involvement (Méndez Fernández and Wagner, 2015), allow the companies to choose themselves which quality criteria to apply (Napier et al., 2009), introduce our treatment to multiple companies (Sommerville and Ransom, 2005) and collect both qualitative perception data and quantitative metrics (Damian et al., 2005). Different from Damian's study (2005), our metrics include quantitative and qualitative data on indirectly related human processes around software development.

There are many other possible metrics. Holm et al.'s systematic literature review (Holm et al., 2015) provides an overview of how previous literature conceptualizes and operationalizes RE. Their examination of 78 studies reveals that in total researchers used 298 dependent variables corresponding to 37 unique classes and they find there is no agreement on how to measure RE success and unclarity in the choice of the variables. However, RE validation studies like ours do perform best: 60% of all dependent variables are of the type *defects found* and the majority of dependent variables include a motivation.

4.5 DISCUSSION AND OUTLOOK

We studied the effect of applying the Grimm Method's QUS framework and the AQUSA tool into existing user story practices through a multiple case study. Although the number of user story quality defects decreased by 43.14% after applying the treatment, participants did not perceive a meaningful change in user story quality. Yet, the respondents agreed that communication frequency and effectiveness improved.

On the negative side, we found no statistically significant difference in communication frequency, communication effectiveness, work quality and work productivity perception between the pre-treatment and experimental periods. Furthermore, our study of the impact on work practices by measuring software process metrics (number of comments, issues, defects, velocity and recidivism rate) did not lead to statistically significant results, perhaps also due to organizational changes between the two periods.

Taking our results into account, we cannot accept our hypotheses H1–4 from Section 4.2. Further investigation is required to obtain more decisive results. However, the results make us hypothesize that improving user stories' intrinsic quality in itself is not essential, but that highlighting quality criteria defects seems to stimulate relevant and meaningful discussion around user stories (a key activity according to Cohn (Cohn, 2004)). For example, the quality criterion *minimal* is seldom resolved by simply removing the text between two brackets, as the in-brackets text is often important: during Grimm Method workshops, defects of this type often indicated an insufficiently refined user story to be further discussed prior to assigning it to the sprint backlog. The consequence of these dynamics is not necessarily a higher quality user story or a direct increase in productivity, but a team that may more quickly agree upon the requirements.

Another explanation for the outcome is that the treatment is wrong, incomplete or even overcomplete. Although the QUS framework describes 13 quality characteristics, the individual relevance is unknown. Replacing or removing some criteria could improve the results. Also, in this study we focused on just the 5 (out of 13) characteristics that the AQUSA tool automatically detects. It would be interesting to conduct a study on all of QUS framework's criteria and to assess their individual impact.

Threats to validity. Multiple human factors should be considered and many aspects of the study design are hard to control for. Two important internal validity threats are the *Hawthorne Effect* and *good participant response bias* which causes participants that are aware of being observed to (sub)conciously modify their behavior. Related is the first *confounding* variable: simply instructing participants to pay attention to user story quality when creating them could be enough to explain the change in intrinsic user story quality. Additionally, there is a risk of *regression toward the mean*: if the user stories' quality was very poor, they would have improved regardless of the applied quality criteria. Although we tried to control for the latter two validity threats by selecting case companies with multiple years of user story experience, their relevance persists due to the absence of control groups. Note that all four

of these threats could explain why we did not detect statistically significant positive changes, yet participants do agree that communication frequency and effectiveness improved after applying the treatment.

Our application of the treatment and measurement of the results introduce two other validity threats. After attending the training, the participants could have attempted *hypothesis guessing*: knowing the desired result changes their actions. In a larger scale study it is possible to control for this threat by leaving participation in the study open ended and not informing participants of data collection. A large scale study also allows taking into account other potentially confounding variables such as the number of people in the project or the company size. For example, in the large eCommerce company, frequent team composition changes could influence the team's acceptance of new methods. Furthermore, there is an evident *history* threat: events outside of the researchers' control affect the outcome of the results. In the chaotic environment of software companies, many unforeseen events occur over a two month period that can affect the collected data. Although unavoidable in empirical research, substantially increasing the scale of the study is likely to reduce the impact of on the data.

Finally, there is a potential *bias in the experimental design*: the number of comments in Jira is not sufficient to measure the intended effect of increasing communication. In agile software development, much of the communication is verbal and informal. Unfortunately, it is extremely hard to accurately measure and record verbal communication.

Future work and outlook. The mildly positive participant perception is not confirmed by software development process metrics. As in similar studies (Méndez Fernández and Wagner, 2015; Sommerville and Ransom, 2005), a large scale replication is necessary to generalize our conclusions, also to reduce threats to validity by controlling for confounding variables as company and project size. Additionally, we want to study the implications of specific quality criteria; what are the consequences of a minimality or uniformity defect in isolation? Also, we aim to devise tools that learn how to suggest improvements based on the most common resolution strategies.

Finally, we invite the RE community to undertake similar studies with the aim of measuring the *actual* effect of RE methods on work practices; our mixed results emphasize the importance of replication studies but also highlight some of the problems that other researchers could encounter in the evaluation of their own solutions.

Extracting Conceptual Models from User Stories with Visual Narrator

5

Extracting conceptual models from natural language requirements can help identify dependencies, redundancies and conflicts between requirements via a holistic and easy-to-understand view that is generated from lengthy textual specifications. Unfortunately, existing approaches never gained traction in practice, because they either require substantial human involvement, or they deliver too low accuracy. In this paper, we propose an automated approach called Visual Narrator based on natural language processing that extracts conceptual models from user story requirements. We choose this notation because of its popularity among (agile) practitioners and its focus on the essential components of a requirement: Who? What? Why? Coupled with a careful selection and tuning of heuristics, we show how Visual Narrator enables generating conceptual models from user stories with high accuracy. Visual Narrator is part of the holistic Grimm method for user story collaboration that ranges from elicitation to the interactive visualization and analysis of requirements.

This work was originally published as:

G. Lucassen, M. Robeer, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Extracting conceptual models from user stories with visual narrator. *Requirements Engineering*, 22 (3):339–358, 2017c

5.1 INTRODUCTION

The software industry commonly uses natural language (NL) notations to express software requirements (Neill and Laplante, 2003), with NL being employed by over 60% of practitioners (Kassab et al., 2014). With the increasing adoption of agile development practices such as Scrum, the semi-structured NL notation of *user stories* is gaining momentum (Kassab, 2015; Lucassen et al., 2016a).

NL requirements are easy to understand because they employ the very same language that we use to communicate with others. Nevertheless, NL suffers from several drawbacks too. The ambiguity of words and sentences is a well-known and widely studied problem that results in different interpretations of the same text. See Berry et al. (Berry et al., 2001) for an authoritative review. In this paper, we focus on another difficult problem: the identification and exploration of the key entities and relationships in a large set of requirements. Our work is intended to support the detection of dependencies between requirements, redundancies, and inconsistencies.

Our baseline consists of the existing literature on deriving conceptual models from NL requirements (Omar et al., 2004; Du and Metzler, 2006; Harmain and Gaizauskas, 2003; Hartmann and Link, 2007). However, we go beyond the limitations of these inspiring techniques, which either (i) *require human supervision* to appropriately tag the entities and relationships in the text (Du and Metzler, 2006; Overmyer et al., 2001), or (ii) *have low accuracy*, often due to the ambitious attempt to support arbitrarily complex requirements statements (Harmain and Gaizauskas, 2003; Sagar and Abirami, 2014).

Several studies have shown the added value of conceptual modeling in software development (Davies et al., 2006; Fettke, 2009). Yet, practitioners remain reluctant to adopt these methods. In 2006, just 13% of the Australian Computer Society's 12,000 members indicated an interest in conceptual modeling (Davies et al.). A probable explanation is that the benefits do not outweigh the burden of constructing and maintaining a conceptual model manually.

Our goal is to overcome the limitations stated above and to promote the adoption of conceptual models for discussing about requirements. Our recipe includes three main ingredients: (i) our chosen notation is *user stories*, which are highly popular among practitioners (Kassab, 2015; Lucassen et al., 2016a) and that express concisely the essential elements of requirements (Who? What? Why?); (ii) we minimize human supervision by proposing a *fully automated* software tool; and (iii) we deliver value by focusing on *high accuracy*, and we do so by carefully choosing heuristics that help create a holistic view of

the requirements, and by ignoring those that contribute with too fine-grained details (and are often less accurate).

In previous work (Robeer et al., 2016), we have shown the feasibility of this recipe by introducing the Visual Narrator tool for extracting conceptual models from user stories via NLP. Prior to that, we introduced a conceptual model and an NLP-enabled tool for assisting users in writing high-quality user stories (Lucassen et al., 2015) that obtained promising results (Lucassen et al., 2016b). In this paper, we extend the work in (Robeer et al., 2016) by making two main contributions:

- We combine Visual Narrator with our other NLP tools *AQUSA* and *Interactive Narrator* into the comprehensive Grimm Method for conducting RE with user stories while stimulating the discussion about requirements among team members.
- We make technical improvements to Visual Narrator's algorithms and conduct a new quantitative evaluation with four cases, two of which are completely new, resulting in improved accuracy.

The rest of the paper is structured as follows. Section 5.2 introduces our Grimm Method that combines our research baseline. Section 5.3 reviews 23 heuristics from the literature and selects 11 for use with user stories. Section 5.4 presents the architecture, the main algorithm of the tool and elaborates upon the made technical improvements. Section 5.5 reports on our quantitative evaluation on four data sets from the software industry. We discuss related work in Section 5.6, while Section 5.7 presents conclusions and future directions.

5.2 BASELINE: THE GRIMM METHOD

This paper is part of our ongoing research line on user stories. The premise of our research is the high adoption yet low quality of user stories in industry (Kassab et al., 2014; Lucassen et al., 2016a,b). To improve this situation we focus on fostering deeper understanding of user stories by creating tool-assisted techniques that support practitioners in creating and communicating about high quality user stories (Lucassen et al., 2015, 2016c).

In Section 5.2.1 we present the conceptual anatomy of user stories, while in Section 5.2.2 we introduce the main elements of the Grimm method for conducting requirements engineering (RE) based on user stories.

5.2.1 *Conceptual anatomy of user stories*

Based on previous work (Wautelet et al., 2014; Lucassen et al., 2016b; Cohn, 2004) we define a generic representation of user stories—see the UML class diagram in Fig. 5.1—that dissects a user story into its constituents. User stories follow a standard predefined format (Wautelet et al., 2014) to capture three aspects of a requirement:

1. *Who* wants the functionality;
2. *What* functionality the end users or stakeholders want the system to provide; and
3. *Why* the end users and stakeholders need this functionality (optional).

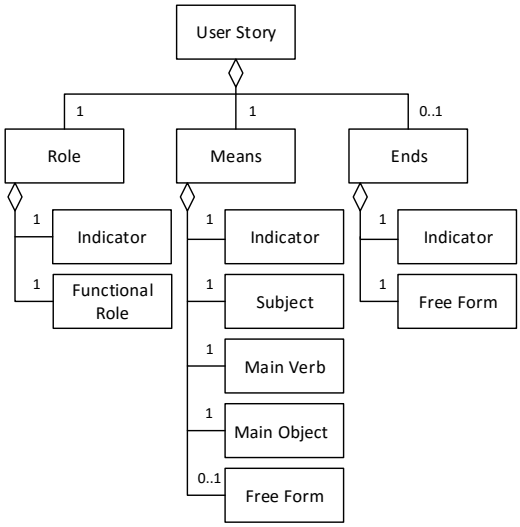


Figure 5.1: Conceptual anatomy of a user story

These three aspects are captured in a simple textual template to form a running sentence. Although many different templates exist, 70% of practitioners use the Connextra template *"As a ⟨type of user⟩, I want ⟨some goal⟩ [so that ⟨some reason⟩]"* (Cohn, 2004).

We distinguish between the *role*, *means* and *ends* parts of a user story. Each of these parts features an *indicator* delimiting the three basic parts of a user story. Jointly, the indicators are the *template* of a user story.

The *role* part encompasses the role indicator and the *functional role*, describing a generalized type of person *who* interacts with the system. When no ambiguity exists, we use the term *role* to denote both the full part and the *functional role*. The *means* consists of a *subject*, a *main object* of the functionality, and a *main verb* describing the relationship between the subject and main object. The main object can be represented explicitly by the direct object or implicitly (e.g., 'I want to log in' actually refers to logging in the *system*). The rest of the *means* can assume too many variations in practice; as such, we do not make any further distinction: words are captured by the *free form* class.

For example, consider the user story *"As a visitor, I want to purchase a ticket so that I can attend the conference"*. Here, 'visitor' is the functional role, 'I' is the subject, 'a ticket' the main object (a *direct* object), and 'purchase' is the main verb linking subject and object. There is no free form part for the means, and the indicators are 'As a', 'I want to', and 'so that'.

Although the ends has a similar structure to the means in this example, this is not always the case (Lucassen et al., 2016b): there are at least three main purposes for having the ends: (1) clarifying the means, (2) referencing another user story, or (3) introducing a qualitative requirement. These functions can be combined for a single user story. This semantic distinction between ends types is beyond the scope of this paper and is left for future work.

5.2.2 *The Grimm Method: Overview and Tooling*

The Grimm method that we propose is our response to the low quality of user stories in industry (Kassab et al., 2014; Lucassen et al., 2016a,b), despite their popularity. Grimm features automated tools that improve the situation (Lucassen et al., 2016b,c). Fig. 5.2 illustrates how Grimm combines our tools in such a way to stimulate discussion around user stories among the stakeholders; this is one of the key objectives of user stories (Cohn, 2004).

In the following paragraphs we elaborate on how each tool contribute to this goal: (1) AQUSA detects quality defects in human-made user stories, (2) Visual Narrator extracts entities and relationships from a user story collection to construct a conceptual model and (3) Interactive Narrator generates specialized views of the conceptual model that facilitate discussions for identifying inconsistencies, dependencies and ambiguities.

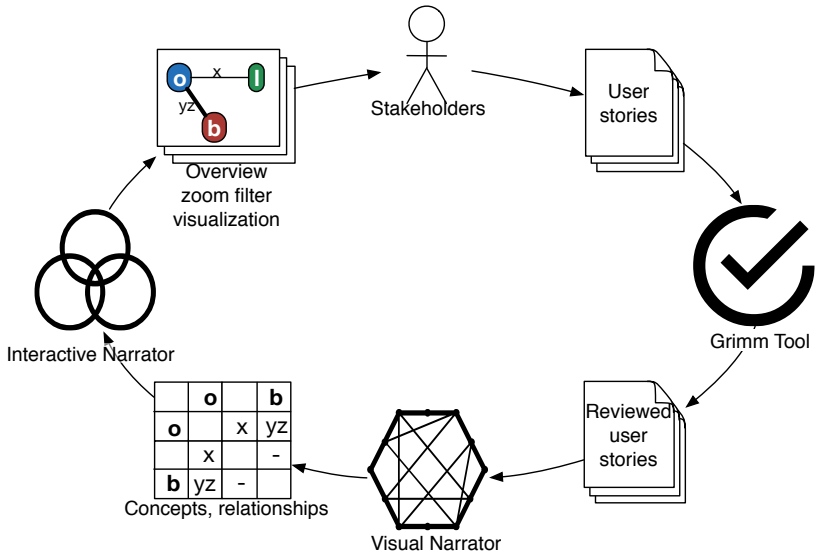


Figure 5.2: The Grimm method for user-story-based RE

AQUSA

Grimm begins when stakeholders formulate some user stories. First, the AQUSA tool (Lucassen et al., 2016b) validates their quality by automatically detecting defects using NL processing techniques. Based on the Quality User Story Framework, AQUSA focuses on those quality criteria can have the potential to be detected with 100% recall. Although this Perfect Recall Condition is theoretically unattainable (Ryan, 1993), AQUSA has proven capable of uncovering up to 90% of *easily preventable defects* in user stories (Lucassen et al., 2016b). Then, a human requirements engineer analyzes the reported errors and take corrective actions when needed. This leads to a revised collection of reviewed user stories, which may involve rephrasing some stories, splitting them, or removing text that is not essential for a user story such as references to design documents.

Visual Narrator

After preprocessing with AQUSA, the user stories can be analyzed further. Visual Narrator applies heuristics from the conceptual model field to extract the relevant entities and their relationships. The result is a comprehensive conceptual model of the entire user story collection. A key use case for this output is to check the completeness and consistency of the entities and

relationships the user stories talk about. For example, one could identify isolated entities that are not connected to others and viewpoints that haven't been fully explored yet such as a role connected to very few entities. The remainder of this paper focuses on explaining Visual Narrator's inner workings and evaluating its accuracy.

Visual Narrator can generate output as a Prolog program (to be used for further automated reasoning) or an OWL 2 ontology. The latter can be used in readily available ontology visualization tools such as WebVOWL (Lohmann et al., 2015) to graphically explore the conceptual model.

Interactive Narrator

Although a preliminary evaluation with practitioners showed the potential of Visual Narrator, the extracted models quickly become too large for human analysts (Robeer et al., 2016). This cognitive overload is a well-known problem for conceptual model comprehensibility (Aranda et al., 2007; Moody, 2009). As a response, we created the so-called *Interactive Narrator* (Lucassen et al., 2016c) that generates specialized views of the conceptual model. Adhering to Shneiderman's Visual Information-Seeking Mantra: "overview first, zoom and filter, then details-on-demand" (Shneiderman, 1996), we combine Visual Narrator's output with other data sources to create specialized views that highlight different user story elements: (1) clustering entities based on state-of-the-art semantic relatedness algorithm Word2Vec (Goldberg and Levy, 2014) and (2) filtering on specific user story details such as roles as well as project management data from issue trackers such as Jira[1].

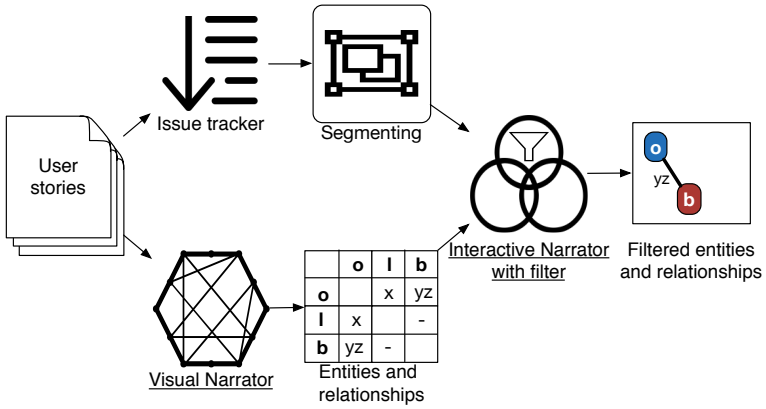


Figure 5.3: Method for filtering a large conceptual model

[1]https://www.atlassian.com/software/jira

As highlighted by our qualitative study via interviews concerning Visual Narrator (Robeer et al., 2016), key requirements for this tool are to help identify and resolve inconsistencies, dependencies (Wang et al., 2014), and redundancies between the requirements. The literature shows that the conceptual model can also play a role in reducing the ambiguity of the requirements (Popescu et al., 2008b). Interactive Narrator is intended as a real-time, modern documentation tool for agile development (Rubin and Rubin, 2010) that fosters and facilitates effective discussion among stakeholders about the software system (to be) (Cohn, 2004). These possible uses are enabled by the key advantage of graphical representations over NL in holistically representing a given domain.

While our initial prototype only supports semantic clustering and zooming[2], we have since built upon our proposal and realized a filtering prototype available on GitHub[3]. Aside from generating views that highlight a specific role or relationship, it supports filtering based on *agile artifacts*. In agile software development, user stories are frequently managed in an issue tracker such as Jira. These types of tools allow the user to organize a user story collection into meaningful chunks: epics, themes and sprints. Interactive Narrator combines this data with entities and relationships extracted using Visual Narrator as shown in Fig. 5.3. By providing the option to select any combination of these, the user can explore specific parts of the system (via epics and themes) or focus on certain development periods (sprints). For example: a user story can be part of the epic 'Presentation' in sprint 6 and simultaneously belong to the theme 'Conference' (see Fig. 5.4).

Each of Interactive Narrator's views zoom in on specific aspects of the system, so as to help end users in identifying inconsistencies, dependencies and ambiguity between requirements (Wang and Wang, 2016; Popescu et al., 2008b). Stakeholders can then use the views as input for collaboratively resolving issues by modifying existing user stories or identifying new ones. This triggers a new method iteration in Fig. 5.2, which repeats until all issues are resolved.

[2]https://github.com/gglucass/Semantic-Similarity-Prototype
[3]https://github.com/Gionimo/VNwebapp

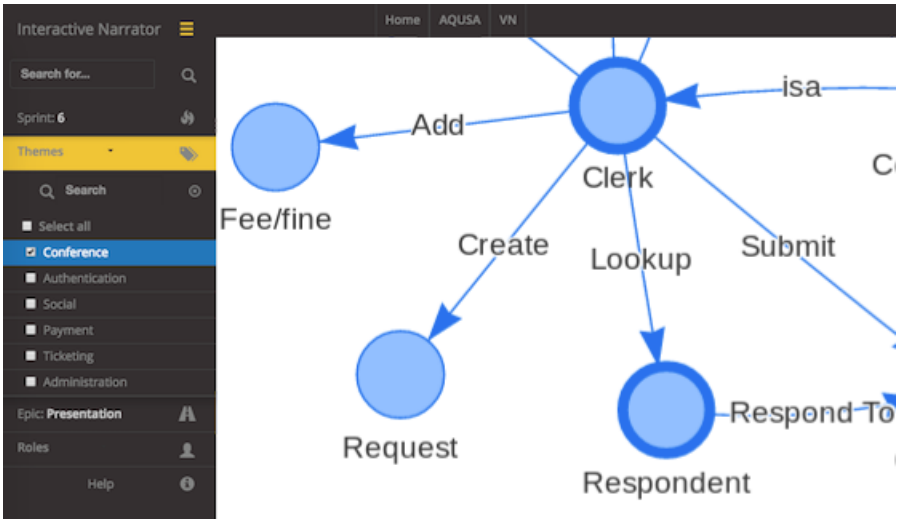


Figure 5.4: Filter example showing entities and relationships for user stories in sprint 6 belonging to theme Conference and epic Presentation

5.3 NLP HEURISTICS FOR USER STORY ANALYSIS

To extract meaningful models from NL requirements, researchers have been proposing *heuristic* rules for the identification of entities and relationships whenever the text matches certain patterns of the given language (usually English). The purpose of this section is to select NLP heuristics that can be effectively employed to derive conceptual models from user stories.

Our literature study on conceptual model generation identified the 23 heuristics shown in Table 5.1. This overview groups the heuristics by the part of a conceptual model they generate: entities, non-hierarchical relationships, hierarchical relationships, attributes, and cardinality. The table presents a simple version of each rule as an implication from a condition (the *head*) to a consequence (the *tail*). As an example, the first entity heuristic should be read as **E1**: "If a word is a *noun*, then it is a *potential entity*."

Table 5.1: Classification of heuristics found in previous literature for conceptual model generation.

ID	Rule head (if)	Rule tail (then)	Source(s)
Entities			
E1	Noun	Potential entity	[1][2][3]
E2	Common noun	Entity	[4][5][6][3]
E3	Sentence subject	Entity	[3][7]
E4	Compound noun	Take compound together to form entity	[8][7]
E5	Gerund	Entity	[4][7]
Non-hierarchical Relationships			
R1	Verb	Potential relationship	[2][7]
R2	Transitive verb	Relationship	[4][1][5][7]
R3	Verb (phrase) linking the subject and an object	Relationship	[1][3][7]
R4	Verb followed by preposition	Relationship including preposition	[6]
R5	Noun-noun compound	Non-hierarchical relationship between prefix and compound	[8]
Hierarchical Relationships			
H1	Verb 'to be'	Subjects are children of parent object	[5][3]
H2	Head of noun-noun compound	IS-A relationship between compound and head	[8]
Attributes			
A1	Adjective	Attribute of noun phrase main	[4][1][5][3][7]
A2	Adverb modifying a verb	Relationship attribute	[4][5]
A3	Possessive apostrophe	Entity attribute	[1][7]
A4	Genitive case	Entity attribute	[6][3][7]
A5	Verb 'to have'	Entity attribute	[9][1][3][7]
A6	Specific indicators (e.g. 'number', 'date', 'type', ...)	Entity attribute	[6]
A7	Object of numeric/algebraic operation	Entity attribute	[4]
Cardinality			
CA1	Singular noun (+ definite article)	Exactly 1	[1][2][3]
CA2	Indefinite article	Exactly 1	[1]
CA3	Part in the form "More than X"	$X..*$	[1][6]
CA4	Indicators *many*, *each*, *all*, *every*, *some*, *any*	??..$*$	[1][6]

1 = (Harmain and Gaizauskas, 2003) 2 = (Meziane and Vadera, 2004) 3 = (Tjoa and Berger, 1993) 4 = (Chen, 1983)
5 = (Hartmann and Link, 2007) 6 = (Omar et al., 2004) 7 = (Sagar and Abirami, 2014) 8 = (Vela and Declerck, 2009)
9 = (Aguado De Cea et al., 2008)

The concise format and structure of user stories implies that not all heuristics are equally relevant. For example, user stories are not meant to include information about attributes or cardinality (Cohn, 2004), thereby making those heuristics poorly relevant for our work. Other heuristics are still ignored by or too difficult for state-of-the-art part-of-speech taggers. For example, the mainstream Penn Treebank tags do not distinguish between gerunds and present participle. This exclusion process results in 11 heuristics that are particularly relevant for generating conceptual models from user stories. We explain and illustrate those 11 heuristics in the following.

5.3.1 *Entities and non-hierarchical relationships*

The most basic heuristics in the literature specify that (1) nouns in a sentence denote an *entity*, and (2) verbs indicate a potential *relationship* (Btoush and Hammad, 2015; Meziane and Vadera, 2004). This prompts us to define the first two heuristics:

E1. *"Every noun is a potential entity."*

R1. *"Every verb is a potential relationship."*

Example A: Consider the user story "As a visitor, I want to create a new account." that comprises two nouns (*visitor* and *account*), and one verb (*create*) when we exclude role and means indicators. Rule E1 specifies to create two entities visitor and account and rule R1 originates a relationship between these entities named as the verb: create(visitor,account).

However, uncritically designating all nouns as entities would result in a conceptual model with superfluous entities. Previous authors have employed the distinction between *proper nouns* and *common nouns* to generalize some of the identified entities as more abstract instantiations (Chen, 1983; Hartmann and Link, 2007; Omar et al., 2004; Tjoa and Berger, 1993). In general, common nouns are entities and proper nouns are instances of these entities that can be disregarded. *Transitive verbs* have a similar function, referencing an object in the sentence. These two phenomena lead to heuristics E2 and R2:

E2. *"A common noun indicates an entity."*

R2. *"A transitive verb indicates a relationship."*

To form relationships between entities, a sentence should contain three components: the subject, the object and the verb (phrase) linking the previous two. The *subject* is certainly essential: in an active sentence, for instance, the subject is the initiator—the so-called *agent*—of the main action performed in the sentence. Therefore, it has its own heuristics:

E3. *"The subject of a sentence is an entity."*

R3. *"The verb (phrase) linking the sentence subject and an object forms the relationship between these two."*

Example B: Let us consider the story "As John the manager, I want to design a website." The sentence comprises two common nouns, one proper noun and one transitive verb. The person *John*—a proper noun—can be generalized to his job description *manager* (E2). Therefore, John is an instance of entity manager. Note that this proper noun defines the entity John because heuristic E3 says that, no matter its type, the subject of a sentence leads to an entity. The transitive verb has *website* as its direct object, and subject *I* which refers to *John* (R2). As we do not know whether the ability to design a website applies to John or to managers in general, we create entity John (E3) with relationship design(John,website) (R3).

Concerning relationships, Omar et al. (Omar et al., 2004) distinguish between two types of verbs that indicate relationships: general transitive verbs and verbs followed by a preposition. These prepositions significantly change the meaning of a relationship, and are therefore captured in a separate heuristic:

R4. *"If a verb is followed by a preposition, then the preposition is included in the relationship name."*

Example C: For the user story "As a visitor, I want to search by category", we first identify the subject *I* (E3). The sentence has no direct object. The preposition *by* (R4) changes the meaning of the relationship from searching something to searching by something. Therefore, we obtain the relationship search_by(I,category). Since *I* refers to the functional role *visitor*, it results in the relationship search_by(visitor,category).

5.3.2 *Compound Nouns*

Compound nouns describe an entity that includes multiple words. Most often these are sequences of nouns or adjectives that precede a noun. To accurately

construct a conceptual model, we consider the whole compound noun as the entity. Compound nouns are known to have many inherent relationships, as there are many ways to combine them (Gagné, 2002). However, extracting this requires the synthesis of lexical, semantic and pragmatic information, which is a complex task (Lapata, 2002) that can hardly lead to accurate results. Therefore, we limit ourselves to a simple heuristic proposed by Vela and Declerck (Vela and Declerck, 2009) by considering only compound nouns of length two:

E4. *"Noun compounds are combined to form an entity."*

R5. *"In noun-noun compounds there is a non-hierarchical relationship between the prefix and compound."*

Example D: The compound noun "event organizer" leads to entity event_organizer (E4) and a "has" relationship has_organizer(event,event_organizer) (R5).

5.3.3 *Hierarchical Relationships*

The ontology generation domain pays special attention to generalization relationships, often referred to as *IS-A* relationships (Hartmann and Link, 2007; Tjoa and Berger, 1993). Tenses of the verb *to be* typically indicate a hierarchical relationship between two entities. The subject of the relationships on the left side of the verb is a specialization of the parent object on the right side of the verb *to be*:

H1. *"The verb 'to be' indicates a hierarchical relationship: the subject is taken as a specialization of the parent object."*

In addition, Vela and Declerck (Vela and Declerck, 2009) note that the nouns in compounds have a generalization relationship. Compound noun entities are a form of a more abstract entity, e.g., database_administrator is a type of administrator. This is captured by the following heuristic:

H2. *"If a noun-noun compound exists, the prefix of the compound is the parent of the compound entity."*

Example E: Consider the user story "As a visitor, I can change my account password." Here, heuristics E4, E5 and H2 apply on noun compound *account password*. We create compound entity account_password (E4), which is a type of password: IS-A(account_password,password) (H2). In addition, we create a 'has' relationship (R5) has_password(account,account_password).

5.4 THE VISUAL NARRATOR TOOL

To automatically extract conceptual models from user stories, we developed the *Visual Narrator* tool that implements the 11 heuristics detailed in Sec. 5.3. Visual Narrator takes a set of user stories as input and generates a conceptual model as output (github.com/MarcelRobeer/VisualNarrator). It is built in Python and relies on the natural text processor *spaCy* (http://spacy.io), a recent proposal in NLP that implements algorithms needing minimal to no tuning and with excellent performance. Additionally, phrasal verb extraction is performed using Li's algorithm (2003).

Our tool only accepts user stories that use the indicators as identified by Wautelet et al. (Wautelet et al., 2014): *As / As a(n)* for the role, *I want (to) / I can / I am able / I would like* for the means, and *so that* for the ends part. Syntactically invalid user stories are not processed; in order to sanitize these stories, analysts should pre-process them using tools such as AQUSA as shown in Fig. 5.2 (Lucassen et al., 2016b).

In addition to generating the conceptual model, Visual Narrator can also generate separate models per role to help analysts focus on an individual role. Furthermore, analysts have the option to fine-tune the sensitivity of the tool: (i) *weights* for each type of entity (role, main object, compound, etc.) can be specified to determine their relative importance, and (ii) a *threshold* can be expressed to exclude from the generated models the least frequent entities by computing a ranking based on frequency and entity weight.

5.4.1 *Architecture*

The conceptual architecture of Visual Narrator is shown in Fig. 5.5 and depicts two main components: (1) the Processor analyzes and parses user stories according to the syntactic model for user stories (Fig. 5.1), while (2) the Constructor creates the actual conceptual model starting from the parsed stories.

First, the Processor analyzes the user story set. The Miner component uses spaCy to parse each user story into tokens, which hold the term itself, its part-of-speech tag and relationships with other tokens. These tokens are stored in the UserStory component and used to infer entities and relationships, and to determine token weights.

Next, the Matrix component removes stop words from the collection of tokens and then attaches a weight to each term, based on the frequency and on the weights that were specified as input parameters. This step results in a *Term-by-User-Story matrix* containing a weight for each term in the individual user

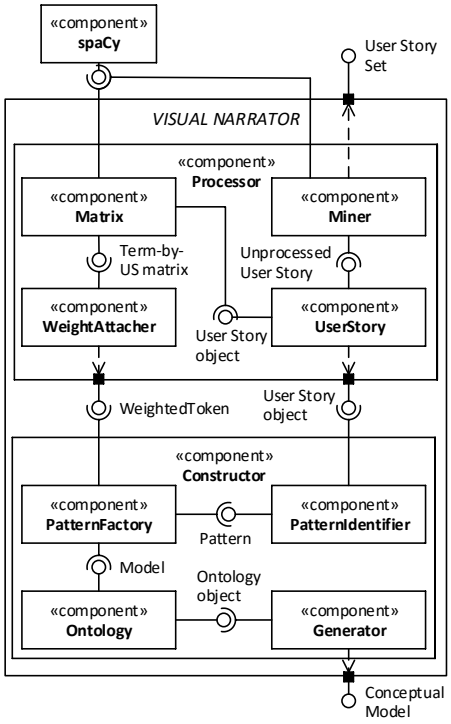


Figure 5.5: Component Diagram of the Visual Narrator tool

stories. The summation of the weights for a given term is added to each token, resulting in a set of WeightedTokens.

The Constructor then generates the conceptual model by further processing the WeightedTokens. This starts in the PatternIdentifier component, which applies the heuristics to identify all patterns in the user story. The PatternFactory component creates an internal conceptual Model based on the patterns and stores it in the Ontology component. Parts of the Ontology are linked to the user story they originated from. Note that the PatternFactory filters out all entities and relationships with a weight below an user-specified threshold. Finally, two Generators output an ontological representation of the Ontology object as an OWL 2 ontology and as a Prolog program.

5.4.2 *Extraction Algorithm*

To extract a conceptual model from user stories, Visual Narrator implements the procedure DeriveCM presented in Algorithm 3. The procedure takes as input a set user stories S as well as empty sets of entities E and relationships R. The procedure then populates E and R while parsing the user stories and applying the heuristics defined in the previous sections.

Algorithm 3 Pseudo-code of DeriveCM that builds a conceptual model by mining stories and applying the heuristics

```
1:  procedure DeriveCM(Stories S, Entities E, Rels R)
2:      ind = ({As a, ...},{I want, I can, ...},{So that})
3:      for each s ∈ S do
4:          (r,m,e) = split-by-indicators(s,ind)
5:          if (r,m,e) == null then continue
6:          else S' = S' ∪ (r, m, e)
7:              for each p ∈ {r, m, e} do
8:                  pt = create-parse-tree(p)
9:                  E = E ∪ find-nouns(pt) [E2]
10:                 E = E ∪ subject-of(pt) [E3]
11:                 for each cn∈comp-nouns(pt) do
12:                     E = E ∪ {cn} [E4]
13:                     R = R ∪ IS-A(cn, prefix(cn)) [H2]
14:                     R = R ∪ has(prefix(cn), cn) [R5]
15:     for each (r, m, e) ∈ S' do
16:         replace-subject(m,r)
17:         if subject-of(e) == "I" then replace-subject(e,r)
18:         for each p ∈ {m, e} do
19:             subj = find-subject(p)
20:             if subj == null then continue
21:             v = find-main-verb(p)
22:             obj = find-direct-object(p) [R2]
23:             if obj == null then
24:                 obj = find-non-direct-object(p) [R3,R4]
25:             if obj == null then continue
26:             if subj, obj ∈ E then
27:                 R = R ∪ v(subj, obj)
28:             else if subj ∈ E ∧ p == m then
29:                 R = R ∪ v(subj, system)
```

The procedure starts in line 2 by defining the valid indicators *ind* for splitting the user stories into role, means, end. The cycle of lines 3–14 excludes syntactically incorrect user stories and creates the set of entities, including hierarchical ones. Every story (r, m, e) is initially split using the indicators (line 4); if this operation fails, the story is discarded and the loop continues to the next story (line 5). If a story is identified, it is added to the set of syntactically valid user stories *S'*.

Every part of a syntactically valid user story (role, means, end) is parsed (line 8). Then, the heuristics to identify nouns (E2) and the subject of the sentence (E3) are executed (lines 9–10); all identified nouns are added to the set of entities E. Lines 11–14 process compound nouns: the compound is added to E according to heuristic E4, a specialization relationship is created linking the sub-entity to the super-entity (H2), and a non-hierarchical "has" relationship links the prefix of the compound to the compound itself (R5).

Lines 15–29 iterate over the set S' of syntactically correct user stories with the intent of identifying relationships where entities in E are linked through associations created by processing the verbs in the user stories. Lines 16–17 modify the means and the end by resolving the pronoun reference: the subject of the means (the "I" of the indicator "I want to") is replaced by the role r; a similar processing applies to ends whose subject is "I" (e.g., "so that I. . . ").

Both means and end are processed in lines 18–29. Subject, main verb and direct object (R2) are identified in lines 19, 21, 22, respectively. If no subject is identified, the algorithm continues to the next user story element: we do not look for verbs when the subject is unclear or nonexistent. If a direct object is not found, an indirect object is searched for applying heuristics R3 and R4 (lines 23–24). If no object is found, the cycle continues to the next user story element (line 25). If both subject and object are in E, a relationship with the name of the verb is created between them (lines 26–27). In case only the subject is in E and we are analyzing the means, a relationship is created from the subject to a special entity called system. For example, the story "As a user, I want to login" would result in a relationship login(user,system). This rule applies only to the means because this part refers to a desired functionality, while the structure of the end is less rigid (Lucassen et al., 2016b).

5.4.3 *Tool Implementation and Improvements*

The aforementioned architecture and algorithm based on the heuristics of Sec. 5.3 capture the design and intention of Visual Narrator. The actual implementation, however, deviates from this in some aspects. It adapts some

of the heuristics and includes a number of techniques to go beyond spaCy's initial Part-of-Speech (PoS) tagging in order to further optimize the results.

We modify E2 by including both common and proper nouns, because proper nouns often refer to domain-specific notions such as the name of a software product or a library. To improve accuracy, we replace the pronoun "I" with the noun they refer to when no ambiguity exists (see Sec. 5.4.2). Effectively, this means Visual Narrator assigns every noun to either a new or an existing entity. We did not implement H1 because the correct application of the heuristics requires a deep understanding of the semantics of the "to be" relationship.

Furthermore, after separating a user story in its role, means and ends parts, Visual Narrator reruns the PoS tagger on each of these shorter parts. Because these partial user story phrases are tagged with higher accuracy we use these as the basis for further analysis. Even if the PoS tagger fails due to missing word classes in a fragmented or grammatically incorrect sentence, Visual Narrator includes fall-back mechanisms to parse the user story part: (i) when no noun is present in the role part the entire text between the two indicators is adopted as the role; (ii) if spaCy does not detect a verb and/or a direct object in the means, Visual Narrator presumes the word directly after the indicator is the verb and assigns the first object in the sentence as the direct object; and (iii) if no object is available at all, it assigns *System* as a default object.

To improve accuracy over our previous work (Robeer et al., 2016), we carefully examined Visual Narrator's output to identify improvement opportunities. This resulted in several minor bug fixes and the introduction of three new features. Two features enhance Visual Narrator's detection of compound nouns, while the third changes how we detect a user story's main object:

- **Amod -** Previously, Visual Narrator relied solely on spaCy's "compound" dependency tag to identify two nouns as a compound noun such as "environment language". SpaCy, however, excludes all compound nouns that comprise a noun adjunct and a noun, instead assigning these the *adjectival modifier* dependency "amod". By including compound nouns with this dependency, Visual Narrator now correctly creates a compound for phrases such as "content types" and "chicken soup".
- **Rightmost child -** There are no agreed upon rules concerning the sequence of a compound noun. Generally speaking, however, the primary noun is the very last one with the nouns before it specializing its meaning: a full moon or a bus stop. Visual Narrator now favors the rightmost child for noun compounds with more than 2 nouns. Parsing the phrase "photo editing tools" now results in compound "editing tools" instead of "photo editing" as in (Robeer et al., 2016).

- **Pobj is main object -** The direct object in a user story is, in general, the object of the role's action. This changes, however, when a user story has a pronoun, adverb or adjective as direct object and includes a *prepositional phrase*. Grammatically, an adverb or adjective cannot be the direct object as they qualify something else. Yet, the data does include this construction because POS taggers parse single adverbs and adjectives as a "noun phrase", resulting in a missing valid "dobj". When a prepositional phrase is present, the main object is found by following prep's "pobj" dependency link as shown in Fig.5.6. Where Visual Narrator would previously extract learn(I,more) from the user story 'I can learn more about the content', the tool now creates the arguably more meaningful relationship learn_more_about(I,content). Note that this new feature results in extra information by including the adverbs in the action, but simply replaces the pronoun with the functional role. We argue, however, that linking the pobj is more informative than keeping the pronoun's context. Consider for example "I register *the site*" versus "I register *myself*" in the case of a user story "I register myself with the site".

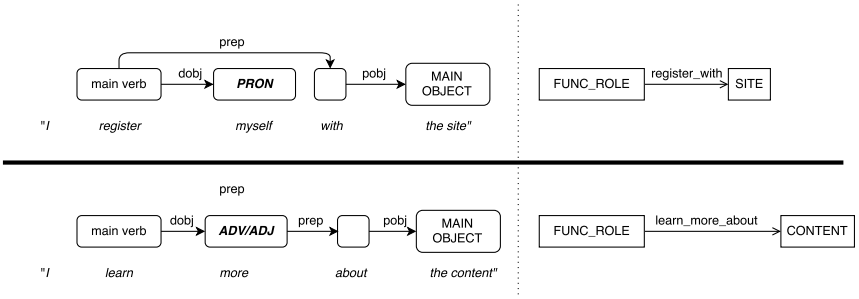


Figure 5.6: Prepositional phrase parsing examples depicting assigning the pobj as main object improvement over (Robeer et al., 2016)

These new features combined with an update to the PoS tagger spaCy means Visual Narrator output is different from the output evaluated in (Robeer et al., 2016). In Sec. 5.5 we report on a thorough new evaluation of Visual Narrator to investigate whether and to what extent these changes result in better recall and precision.

5.5 QUANTITATIVE EVALUATION: ACCURACY

We evaluate the feasibility of our approach and heuristic accuracy by applying Visual Narrator to four data sets from real-world projects: an interactive story telling project by WebCompany, a year of developing CMSCompany's flagship product, a public works project for Michigan State and six months of development for the open source archival software ArchivesSpace. In the following we introduce each in more detail. The data sets and their evaluation documents are available online[4]. An overview of the data sets' characteristics is provided in Table 5.2. The first five rows show the totals per data set: the number of stories n_{us}, of words n_{word}, of entities n_{ent}, of relationships n_{rel} and of words in the user story template n_{tmpl} (e.g., 'so that' means two words). The following three rows indicate averages per user story: number of words $\bar{x}_{word}$, of entities $\bar{x}_{ent}$ and of relationships $\bar{x}_{rel}$. Finally, to measure the user story writing style, the last three rows report on the conceptual density per data set by introducing three metrics for a given set of user stories, i.e., entity density ρ_{ent}, relationship density ρ_{rel} and concept density ρ_{conc}:

$$\rho_{ent} = \frac{n_{ent}}{n_{word} - n_{tmpl}}$$

$$\rho_{rel} = \frac{n_{rel}}{n_{word} - n_{tmpl}}$$

$$\rho_{conc} = \rho_{ent} + \rho_{rel}$$

Looking at the averages, the table reveals that CMSCompany's user stories are long and entity-rich and that despite being shorter, ArchivesSpace's user stories contain more concepts than Michigan State's. Among these four sets the average number of words does not correlate with average entities and average relationships. Looking at the density metrics, we can see how the ArchivesSpace' set is much more conceptually rich than Michigan State's and CMSCompany's; 85% as opposed to 58%. Nevertheless, we could find no correlation between density and the number of words. A detailed study of these metrics and their effect on quality in RE is left for future work.

In Sec. 5.5.1, we determine the accuracy of our implementation of the heuristics (Algorithm 3) by comparing the results of Visual Narrator to a manual labeling of the data sets done by the authors. The outcome of this comparison is encouraging, we obtain accuracies as high as 97% recall and 98% precision with a lower bound of 88% recall and 92% precision. A small but

[4] http://www.staff.science.uu.nl/~lucas001/vn_user_stories.zip

Table 5.2: Lexical characteristics of the evaluation cases.

	Web	*CMS*	*Michigan*	*Archives*
Totals per data set				
n_{us}	79	32	17	49
n_{word}	1549	954	414	783
n_{ent}	400	272	91	294
n_{rel}	247	168	42	156
n_{ind}	645	207	185	261
Averages per user story				
$\bar{x}_{word}$	19.6	29.8	24.4	16.0
$\bar{x}_{ent}$	5.1	8.5	5.4	6.0
$\bar{x}_{rel}$	3.1	5.3	2.5	3.2
Conceptual density per data set				
ρ_{ent}	0.44	0.36	0.40	0.56
ρ_{rel}	0.27	0.22	0.18	0.29
ρ_{conc}	0.71	0.58	0.58	0.85

important accuracy improvement over our earlier work (Robeer et al., 2016). In Sec. 5.5.2, we discuss the limitations of our approach in terms of important entities and relationships that are not recognized due to NLP limitations, our algorithm, or unanticipatable structure of the user stories.

5.5.1 *Heuristics Accuracy*

We evaluate the accuracy of our implemented heuristics by comparing the output of Visual Narrator to manually created golden data sets. In comparison to our evaluation in (Robeer et al., 2016), we have taken a considerably more rigorous approach to constructing these golden data sets in order to ensure their validity: (1) two independent taggers applied Algorithm 3 to the user stories to identify all the entities and their relationships, (2) one tagger compared the two resulting documents and records all discrepancies between them, (3) together, the two taggers resolved the discrepancies by discussing the correct application of Algorithm 3. In our case, there was high agreement between the two taggers, with only 5% to at most 9% discrepancies depending on the data set.

To evaluate our implementation of Algorithm 3 we compare the golden data sets to the analysis by Visual Narrator. Our evaluation has two objectives:

- To determine *quantitatively* to what extent the employed NLP toolkit fails to deliver accurate results due to the difficulty of correctly tagging and parsing sentences.
- To analyze *qualitatively* the limitations of our implementation in terms of important information that is not recognized correctly.

We determine true positive, false positive and false negative user story elements (entities and relationships). As is customary in Information Retrieval reporting, we do not report on true negatives because they shouldn't affect how good or bad the outcome is of your algorithm (Manning et al., 2008):

- *True positive*: the element is identified both by the tool and by the manual analysis.
- *False positive*: the element is identified by the tool but not by the manual analysis.
- *False negative*: the element is not identified by the tool while it was listed in the manual analysis.

Note that multiple heuristics may apply to a given text chunk. For example, three heuristics (see Table 5.1) apply to a compound: C4, R5, and H2. Thus, if the tool misses a compound, it actually generates three false negatives. On the other hand, if the tool and the manual analysis match, three true positives are generated. Moreover, the incorrect identification of a relationship (e.g., see(visitor,content) instead of see(visitor,display_name)) results in both a false positive (the incorrect relationship) and a false negative (the missed out relationship).

We report on accuracy in two ways: (i) on individual user stories, by aggregating the number of true positives, false negatives and false positives for entities and relationships; and (ii) on the obtained conceptual model, by comparing the manually created one against the generated one.

Making this distinction is important when an unrecognized entity appears many times: consider, for example, a compound noun (C4, H2, R5) that appears in 20 different user stories in a data set. This would imply 20 false negative entities and 40 false negative relationships (20 times H2 and 20 times R5), while the resulting conceptual model—the actual artifact that we propose for the stakeholders to use in their discussion—would only miss one entity and two relationships.

Tables 5.3–5.10 report the results using the same format. They have three macro-columns: entities, relationships, and overall (entities+relationships).

Each macro-column has three sub-columns to denote true positives (TP), false positives (FP) and false negatives (FN). The rows indicate the number of instances (*Inst.*), i.e., the number of identified and missed out entities and relationships; the percentile splitting of TP, FP, and FN (%), the precision (PRC), the recall (RCL) and the weighted harmonic mean of PRC and RCL using the F_1 score (Manning et al., 2008) (F_1).

WebCompany

This data set comes from a young company in the Netherlands that creates tailor-made web business applications. The team consists of nine employees who iteratively develop applications in bi-weekly Scrum sprints. *WebCompany* supplied 98 user stories covering the development of an entire web application focused on interactive story telling that was created in 2014. 79 of these 98 user stories were syntactically correct, usable and relevant for conceptual model generation (Lucassen et al., 2016b). Part of the generated conceptual model is shown in Fig. 5.7.

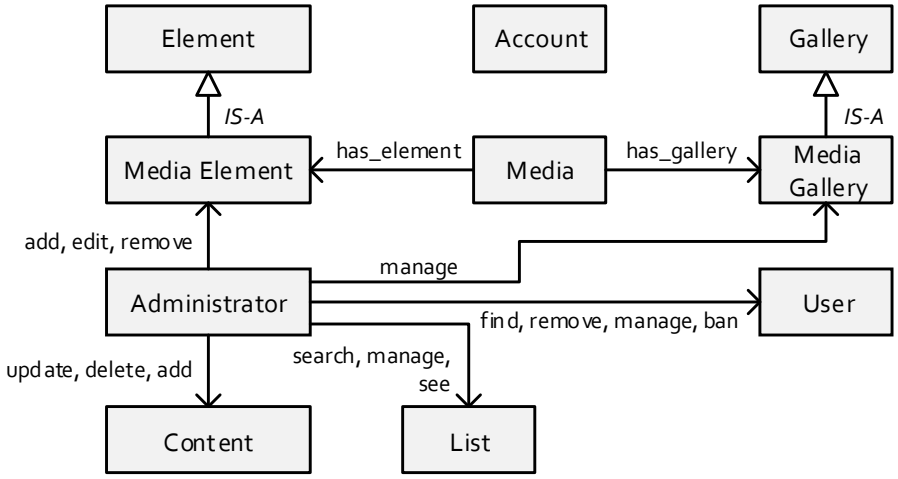


Figure 5.7: Partial conceptual model for WebCompany's Administrator role based on Visual Narrator output

Table 5.3: Accuracy of WebCompany's individual user story analysis (N=79).

	Entities			*Relationships*			*Overall*		
	TP	*FP*	*FN*	*TP*	*FP*	*FN*	*TP*	*FP*	*FN*
Inst.	393	10	7	230	10	17	618	20	24
%	95.9	2.4	1.7	89.3	4.0	6.7	93.4	3.0	3.6
PRC	97.5			95.7			96.9		
RCL	98.3			93.0			96.3		
F_1	97.9			94.4			96.6		

Table 5.4: Accuracy of the generated conceptual model for WebCompany.

	Entities			*Relationships*			*Overall*		
	TP	*FP*	*FN*	*TP*	*FP*	*FN*	*TP*	*FP*	*FN*
Inst.	102	5	5	149	10	15	251	15	20
%	91.1	4.5	4.5	85.6	5.7	8.6	87.8	5.2	7.0
PRC	95.3			93.7			94.4		
RCL	95.3			90.9			92.6		
F_1	95.3			92.3			93.5		

The accuracy results of the individual user story analysis shown in Table 5.3 are positive. The overall precision and recall are 96.9% and 96.3%. The accuracy is higher for entities than for relationships; for entities, precision and recall are approximately 98%, while for relationships precision is 95.7% and recall is 93.0%. This happens because identifying a relationship depends on correctly identifying both the relationship name as well as its source and target entities.

The accuracy of the generated conceptual model (Table 5.4) is also very good, although a bit less accurate than that of the individual user story analysis: overall precision is 94.4% and recall is 92.6%. Interestingly, despite the large overlap in individual errors, the drop in recall and precision are comparable for both entities and relationships (95.3% for both vs. 93.7% precision and 90.9% recall).

Overall, Visual Narrator's output for the *WebCompany* data set is highly accurate. A closer examination of the false positives and false negatives reveals that there are two main causes for errors: (i) *human error* such as using *its* instead of it's confuses spaCy, and (ii) *ambiguous or complex phrases* that the NLP tooling fails to correctly identify, such as *up to date* erroneously resulting in a entity *date*.

CMSCompany

A data set from a mid-sized software company located in the Netherlands with 120 employees and 150 customers. They supplied 34 user stories for a complex CMS product for large enterprises; those stories represent a snapshot of approximately a year of development in 2011. Fig. 5.8 shows a partial conceptual model. The data set of *CMSCompany* included 32 syntactically correct user stories. Despite the smaller size, this data set is particularly interesting due to the use of lengthy user stories with non-trivial sentence structures like: *"As an editor, I want to search on media item titles and terms in the Media Repository in a case insensitive way, so the number of media item results are increased, and I find relevant media items more efficiently".*

Table 5.5: Accuracy of CMSCompany's individual user story analysis (N=32).

	Entities			*Relationships*			*Overall*		
	TP	*FP*	*FN*	*TP*	*FP*	*FN*	*TP*	*FP*	*FN*
Inst.	259	9	13	145	12	23	404	21	36
%	92.2	3.2	4.6	80.6	6.7	12.8	87.6	4.6	7.8
PRC	96.6			92.4			95.1		
RCL	95.2			86.3			91.8		
F_1	95.9			89.2			93.4		

Table 5.6: Accuracy of the generated conceptual model for CMSCompany.

	Entities			*Relationships*			*Overall*		
	TP	*FP*	*FN*	*TP*	*FP*	*FN*	*TP*	*FP*	*FN*
Inst.	129	5	7	112	13	23	241	18	30
%	91.5	3.5	5.0	75.7	8.8	15.5	83.4	6.2	10.4
PRC	96.3			89.6			93.1		
RCL	94.9			83.0			88.9		
F_1	95.6			86.2			90.9		

Table 5.5 presents the results of the analysis of individual stories. The accuracy for the CMSCompany data set is still high, despite the high complexity of its user stories in terms of average words, entities and relationships per user story as shown in Tab. 5.2. The overall precision and recall for individual user stories are 95.1% and 91.8%. This lower accuracy is mostly due to the 86.3% recall for relationships and in smaller part its 92.4% precision. Although the identification of entities also shows lower accuracy (precision 96.6% and recall 95.2%) the decrease is small considering the high user story complexity.

Table 5.6 reports on the accuracy of the generated conceptual model. The less accurate parsing is also reflected here with a less accurate conceptual model: overall precision and recall are 93.1% and 88.9%. While accuracy for entities is nearly identical to in the individual analysis case, the precision and recall for relationships drop to 89.6% and 83.0%.

The main determinant of this performance is the existence of hard-to-process compound nouns that include an ambiguous term such as "content" or "flash" which could also be an adjective or verb. In the worst case, Visual Narrator identifies the wrong compound (C4) and its relationships (H2, R5). As stated earlier, a missed out compound also implies missing out the relationships when the compound is a direct object (R2) or a non-direct object (R3 or R4).

Figure 5.8: Partial conceptual model for CMSCompany's Marketeer role in the conceptual model visualization tool Interactive Narrator

Michigan State

These user stories are extracted from a State of Michigan Enterprise Procurement document. This publicly available document created by the Department of Technology, Management, and Budget in Lansing, Michigan describes a *Statement of Work* concerning the scope and definition of the professional services to be provided by a contracting company: Accela, Inc. For our analysis, we extracted 27 user stories for a complaints system. Although these user stories are concise and obviously written by English native speakers, Visual Narrator could only parse 17 out of 27 user stories, of which Fig. 5.9 shows a partial model. The non-parsed user stories contain minor well-formedness issues that could quickly be resolved by preprocessing with AQUSA (Lucassen et al., 2016b).

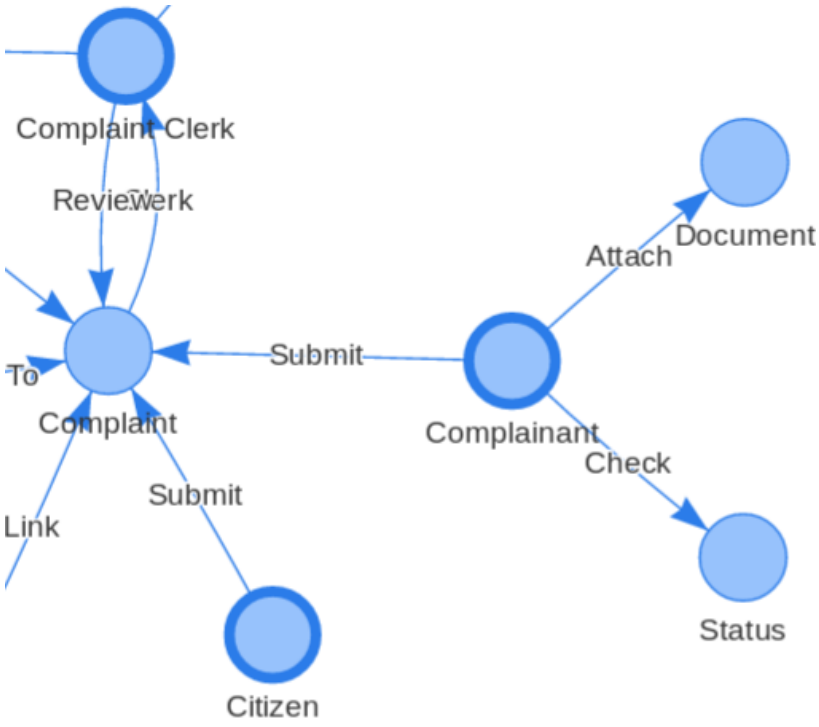


Figure 5.9: Partial conceptual model for Michigan State's Complaint concept in Interactive Narrator

Table 5.7: Accuracy of Michigan State's individual user story analysis (N=17).

	Entities			*Relationships*			*Overall*		
	TP	*FP*	*FN*	*TP*	*FP*	*FN*	*TP*	*FP*	*FN*
Inst.	91	2	0	40	2	2	131	4	2
%	97.8	2.2	0	90.0	4.5	4.5	95.6	2.9	1.5
PRC	97.8			95.2			97.0		
RCL	100.0			95.2			98.5		
F_1	98.9			95.2			97.8		

Table 5.8: Accuracy of the generated conceptual model for Michigan State.

	Entities			*Relationships*			*Overall*		
	TP	*FP*	*FN*	*TP*	*FP*	*FN*	*TP*	*FP*	*FN*
Inst.	46	2	1	38	1	2	84	3	3
%	93.9	4.1	2.0	92.7	2.4	4.9	93.3	3.3	3.3
PRC	95.8			97.4			96.6		
RCL	97.9			95.0			96.6		
F_1	96.8			96.2			96.6		

Despite the small magnitude, the data set is interesting for its extreme simplicity and consistent structure. Although its 5.4 average entities per user story is actually slightly higher than *WebCompany*'s 5.1, the accuracy results shown in Table 5.7 for individual stories push the boundary of what Visual Narrator can achieve: 97% precision and 98.5% recall. One reason for this high accuracy is the inclusion of just 6 compound nouns. In total, there are four errors. Two wrongly identified entities lead to a 98% precision and 100% recall for entities, and two mistakenly generated relationships result in identical precision and recall of 95.2%. Due to the low number of errors, the accuracy of the generated conceptual model is nearly identical: 96.6% precision and recall overall, 95.8% precision and 97.9% recall for entities and 97.4% precision and 95.0% recall for relationships (Table 5.8). Of all the errors in the conceptual model, 4 errors are the consequence of mistakes by the NLP tooling, while the remaining 2 emerge because Visual Narrator does not include personal pronouns in the conceptual model even when they are the subject (with the exception of "I", which is replaced by the referential element as per line 17 in Algorithm 3).

ArchivesSpace

ArchivesSpace[5] is an open source software product created by archivists such as those of the British Royal Archive. The user stories of this project are available online[6]. For our analysis, we extracted 56 user stories that span development from the start of the project in August 28, 2012 until February 28, 2013. Out of those stories, 49 are syntactically correct. The user stories in this collection are quite peculiar: all of them omit the *so that* part of the Connextra template (Cohn, 2004), but many user stories contain unnecessarily capitalized words, compound nouns and idiosyncratic phrases such as "... edit for (Accession | Resources) before ...".

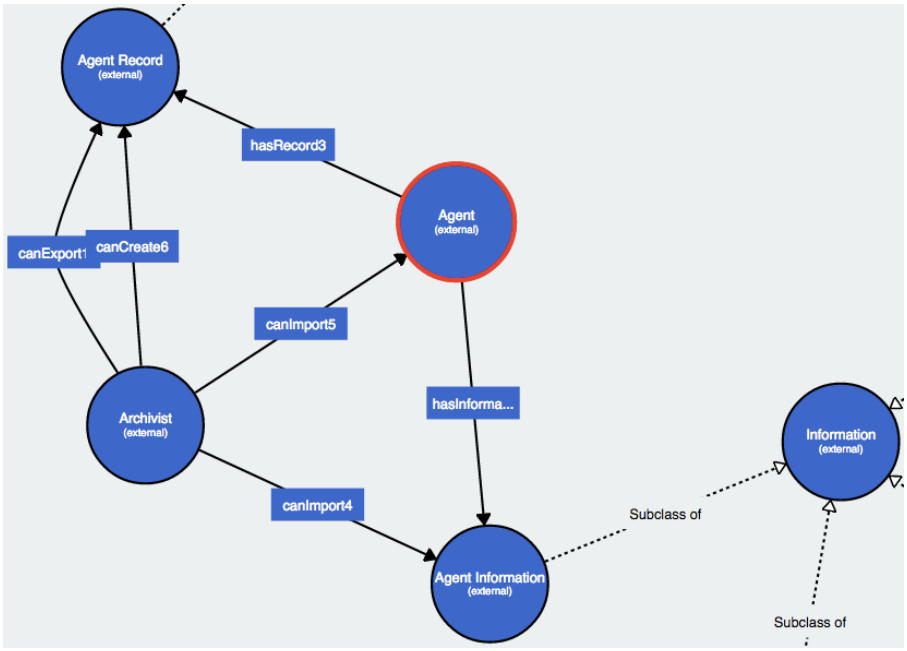


Figure 5.10: Partial conceptual model for ArchivesSpace's Agent concept using the publicly available visualization tool WebVOWL

This non-standard approach has a substantial impact on the analysis and the conceptual model (Fig. 5.10). Despite omitting the *so that*, Visual Nar-

[5]http://www.archivesspace.org/
[6]archivesspace.atlassian.net

rator's accuracy is lowest on the ArchivesSpace data set. For individual user stories, the overall precision is 92.3% and recall is 88.4%. For relationships in particular, precision is 87.6% and recall is 81.4%. Likewise, accuracy in identifying entities is the lowest out of all four sets: 94.8% precision and 92.2% recall.

Table 5.9: Accuracy of ArchivesSpace's individual user story analysis (N=49).

	Entities			*Relationships*			*Overall*		
	TP	*FP*	*FN*	*TP*	*FP*	*FN*	*TP*	*FP*	*FN*
Inst.	271	15	23	127	18	29	398	33	52
%	87.7	4.9	7.4	73.0	10.3	16.7	82.4	6.8	10.8
PRC	94.8			87.6			92.3		
RCL	92.2			81.4			88.4		
F_1	93.4			84.4			90.4		

Table 5.10: Accuracy of the generated conceptual model for ArchivesSpace.

	Entities			*Relationships*			*Overall*		
	TP	*FP*	*FN*	*TP*	*FP*	*FN*	*TP*	*FP*	*FN*
Inst.	137	6	10	116	17	23	253	23	33
%	89.5	3.9	6.5	74.4	10.9	14.7	81.9	7.4	10.7
PRC	95.8			87.2			91.7		
RCL	93.2			83.5			88.5		
F_1	94.5			85.3			90.0		

Table 5.10 reports on the accuracy of the conceptual model. Again, the accuracy results are the worst out of all four data sets: overall precision is 91.7% and recall is 88.5%. Remarkably, however, the precision of entities and recall of both entities and relationships in the conceptual model is superior to individual user story analysis. The reason is that some relationships were missed out in some user stories, but recognized correctly in others, depending on the complexity of the sentence.

The majority of incorrectly identified relationships and entities are due to the (non-)identification of compound nouns. For example: the text chunk "(Accession | Resources)" resulted in the creation of the compound noun / *Resources*, while Visual Narrator did not identify a compound noun for the jargon term *Subject heading*.

5.5.2 *Analysis and Discussion*

The changes described in Sec. 5.4.3 generally improved the accuracy of Visual Narrator over our previous results (Robeer et al., 2016) as highlighted by the bar charts in Figs. 5.11, 5.12, 5.13, 5.14. For the WebCompany case, the average precision for entities and relationships actually dropped by 1%. In exchange, however, recall improved substantially. Entity recall for individual analysis and the generated model improved by 2% and 4%, while relationship recall improved by 6% and 5%. The improvement for CMSCompany is even better, although there were more improvement opportunities. Average precision improved by 1.5%. Entity recall improved by 3% and 4%, whereas relationship recall improved by 11% and 3%. WebCompany's overall accuracy raised from 97.9% precision and 92.9% recall to 96.9% and 96.3% for individual analysis, and from 96.2% precision and 88.2% recall to 94.4% and 92.6% for the model. CMSCompany's overall accuracy changed from 93.8% precision and 86.0% recall to 96.1% and 91.8% for individual analysis, and from 91.9% precision and 85.3% recall to 93.1% and 88.9% for the model.

The simple and semi-structured template of user stories makes them an ideal candidate for NLP analysis as demonstrated by the high accuracy of Visual Narrator. Problems arise when, as in the second and fourth case study, people deviate from the basic format and formulate complex requirements that go beyond the purpose of the user story template. This had a substantial impact for the ArchivesSpace set in particular, with overall accuracies of just 92% recall and 88.5% precision for both individual user story analysis and generated conceptual model. Conversely, the Michigan State user stories are parsed with the highest accuracy thanks to their simplicity and consistency. In the following, we present some key challenges concerning NLP processing for user stories that our evaluation revealed.

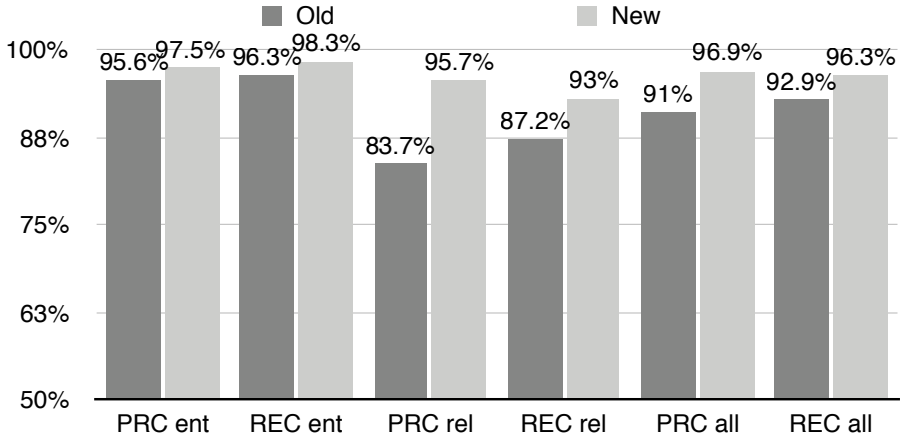


Figure 5.11: Web individual PRC and RCL change

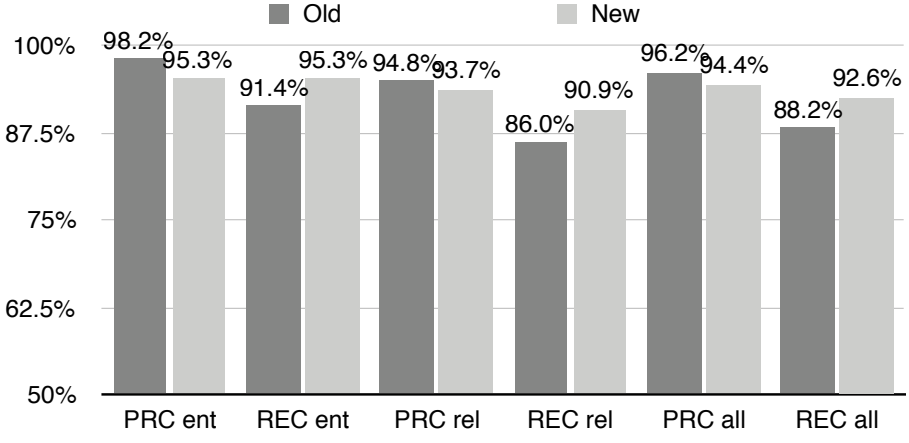


Figure 5.12: Web set PRC and RCL change

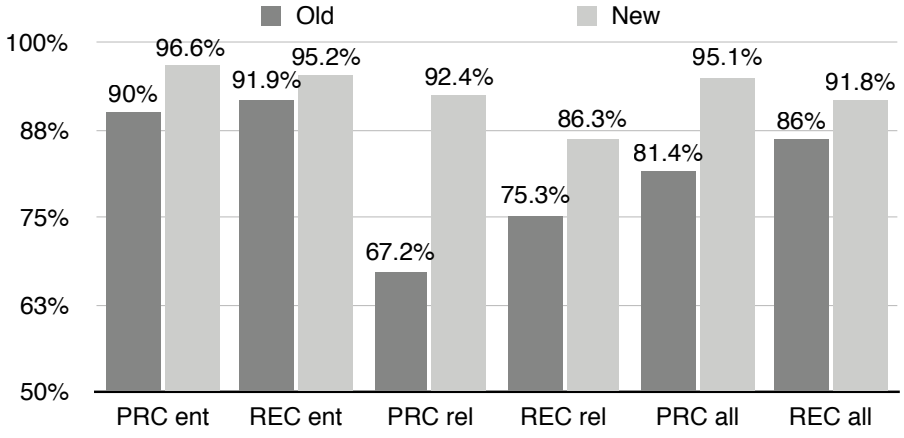


Figure 5.13: CMS individual PRC and RCL change

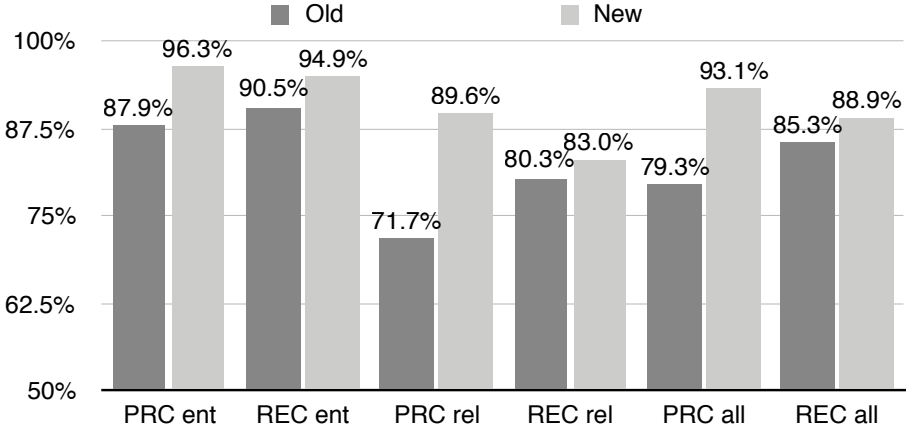


Figure 5.14: CMS set PRC and RCL change

Compounds are difficult to identify correctly. Their proper identification depends on their position within a sentence. For example, the spaCy tagger finds the compound events_section in the sentence "I keep the events section", but misses it in the sentence "I can keep the events section". This is due to the fact that modern NLP taggers are statistical and try to find the most probable tagging in many situations (Manning, 2011). Furthermore, we do not yet support compounds that consist of three or more nouns, such as "profile page statistics". Note, however, that these are often grammatically complex structures with ambiguous semantics: in the example, do "page statistics" refer to the "profile", or do "statistics" refer to the "profile page"? While a part of speech tagger is capable of recognizing "profile page statistics" as a three-word compound, it is unable to disambiguate it semantically.

Verbs may be difficult to link to the proper object. For example, for *CMSCompany*, our implementation identifies the relationship avoid(marketeer,redirects) in the sentence "Marketeer can avoid duplicate content easily without having to set permanent redirects", instead of avoid(marketeer,content). Although spaCy is capable of correctly parsing this sentence, this error occurs due to our simplification of the sentence which replaces "Marketeer can" with "I". In this case spaCy incorrectly tags the ambiguous adjective "duplicate" as a verb and the equally ambiguous noun "content" as an adjective. In future work, we will re-evaluate whether a different approach to simplification is necessary.

Furthermore, while we focused on linking the subject to the main object of the sentence through the verb, there are also sentences requiring **higher arity relationships**. For instance, the sentence "The user can add new elements to the gallery" would require the creation of an n-ary association add-something-to tying together the entities user, element and gallery.

Conjunctions are also a challenge. For the time being, we decided to overlook them, guided by our conceptual model of user stories that calls for their atomicity and minimality (Lucassen et al., 2016b). However, our second and fourth case study include multiple examples of stories that violate these principles, using the conjunction "and" (but also logics-inspired operators such as "|") to specify multiple requirements or expressing conditions using the conjunction "when".

Additionally, we currently **exclude adjectives and adverbs from the conceptual models**. This prevents us from identifying specializations and quality requirements. For example, "external link" is an adjective-noun compound that could induce an attribute or specialization of link. Also, adverbs that qualify verbs ("easily", "intuitively", "faster") could lead to more accurate

relationships that indicate the qualities for the system to comply with. Future work should study whether or not tagging these additional components would produce *more useful* conceptual models: the gained completeness may result in less accuracy, and the trade-off has to be investigated empirically in industry.

Finally, **the human element is difficult to plan for**. A substantial portion of the errors in the WebCompany case are caused by grammatical errors or misspellings. The lower accuracy of the ArchivesSpace case can partly be attributed to the usage of the logical OR sign |. Preprocessing these types of issues would further improve the results of Visual Narrator. An interesting direction is to provide interactive support to stakeholders expressing these requirements, e.g., via a grammar checker or AQUSA (Lucassen et al., 2016b).

5.5.3 *Threats to validity*

External validity threats reduce the generalizability of the results. To obtain even more reliable accuracy results requires substantially larger evaluations. Moreover, our data sets were obtained through convenience sampling from industry contacts and online searches. It is unclear whether these data sets are representative of user stories in general.

Construct validity threats are about the degree to which a test measures what it intends to measure. Measuring accuracy requires understanding the correct interpretation of a sentence; it is well known that NL is *inherently* ambiguous. To reduce the risks, we limited ourselves to the more objective criterion of compliance with the algorithm, instead of the general case of recognizing all entities and relationships. In fact, we argue that comparing Visual Narrator output with theoretically ideal output is a more objective evaluation than comparing with a human's subjective recognition of all relevant concepts and relationships in a text.

Nevertheless, by taking this approach we are unable to empirically evaluate the validity of the algorithm itself. While our precision and recall for detecting what we want to detect is high, we do not know whether what we want to detect is actually useful. In earlier work, we conducted a preliminary evaluation with two practitioners who indicated Visual Narrator output is usable and promising (Robeer et al., 2016). However, a larger scale evaluation with practitioners is necessary to determine the subjective validity of the algorithm.

Internal validity threats focus on how the experiments were conducted. The golden data sets were tagged and applied manually. Although multiple human taggers were used, there is still a low risk that human error caused incorrect analysis.

5.6 RELATED LITERATURE

5.6.1 *User stories*

The growing popularity of agile development practices such as Scrum leads to a continuously increasing adoption of user stories (Kassab, 2015; Wang et al., 2014). Although research interest is slowly growing, the currently available literature is limited. Arguably, the first literature on incorporating stories into requirements is within the scenario-based requirements engineering domain in the 1990s (Holbrook, 1990). The earliest research work on *user* stories proposes their use as the first artifact for describing interactions containing a role, an action and some object (Imaz and Benyon, 1999). For implementation purposes the authors propose transforming user stories into a more formal notation such as use cases.

The majority of research in the field attempts to create methods and tools that support or improve user story practice. In 2002, Rees proposed to replace the pen-and-card approach for writing user stories with a software tool called DotStories (Rees, 2002). Today's popular agile project management tools such as Jira and Trello are all built around the skeuomorph index card metaphor introduced with DotStories.

As of 2015 academic interest in user stories is renewed, leading to a variety of research initiatives. Recent studies predominantly investigate how to connect and/or integrate user stories with different modeling techniques. For example, Trkman et al. propose a method to associate user stories with business process modeling activities (Trkman et al., 2016). They find that undergraduate students better understand user stories' execution order and integration dependencies when business process models are available.

A variety of studies propose new ways to employ or work with user stories to achieve some goal. For example, Barbosa et al. demonstrate how to identify semantically duplicate user stories by applying well known similarity measures. They test their approach with 3 case sets, automatically identifying up to 92% of duplicates (Barbosa et al., 2016). Observing that practitioners encounter difficulties incorporating user experience concerns into user stories, Choma et al. propose extending the Connextra template (Choma et al., 2016). Although their case company fully adopted this new template, more research is necessary to evaluate the added benefit.

Other studies investigate some aspect of user stories in practice. Dimitrijevic et al. qualitatively compare five agile software tools in terms of their functionality, support for basic agile RE entities and practices, and user

satisfaction. They conclude that basic functionality is well supported by tools, but that user role modeling and personas are not supported at all (Dimitrijević et al., 2015). Soares et al. investigate the link between user stories and documentation debt, finding that the low level of detail of user stories is the primary reason for difficulties (Soares et al., 2015).

Finally, two authors take a different approach to ours for transforming a user story set into a visual representation. Wautelet et al. propose a method that results in Use-Case Diagrams (Wautelet et al., 2016a). The authors demonstrate a CASE-tool that automates their approach and allows end-users to iteratively improve and extend the output. In another project (Wautelet et al., 2016b), these same authors propose a method for mapping user stories to agent-based software architecture using i* (Yu, 1996; Dalpiaz et al., 2016). Similarly, the US2StarTool (Mesquita et al., 2015) derives skeletons of i* goal models from user stories.

Whereas our approach employs NLP to extract all entities and relationships from a user story, these tools strictly map the entire action text to a task and the benefit to a goal. While employing the i* and use case notations enable these models to be more expressive than ours, they also require a human to actually construct a model by assembling the extracted parts. Visual Narrator's fully automatic model generation allows all stakeholders to quickly get an understanding of the software system's functionalities and partake in relevant and meaningful discussion around the user stories. A key activity according to Cohn (Cohn, 2004).

Aside from facilitation communication, Visual Narrator output functions as the foundation for employing user stories to achieve other goals. For example, combining the extracted entities with semantic similarity calculations enables grouping user stories in clusters as we do in (Lucassen et al., 2016c) or identifying semantically duplicate user stories similar to the work in (Barbosa et al., 2016). Another application we are currently exploring is reconciling Visual Narrator output with class diagrams by automatically connecting an extracted entity to the source code it executes via automated acceptance tests.

5.6.2 *NLP for RE: Extracting models from requirements*

Historically the final frontier of RE has been applying natural language processing. Nowadays, the ambitious objective of full automation is considered unattainable in the foreseeable future (Ryan, 1993; Berry et al., 2012). Therefore, RE research has applied NLP to specific use cases. Berry et al. (2012) categorizes the fundamental approach of all NLP RE tools into four types:

1. Finding defects and deviations in natural language (NL) requirements document;
2. Generating models from NL requirements descriptions;
3. Inferring trace links between NL requirements descriptions;
4. Identifying the key abstractions from NL documents

In (Lucassen et al., 2016b) we provide an overview of contemporary tools in NLP for RE and introduce the AQUSA tool for automatically detecting quality defects in user stories. Furthermore, observing that no objective comparison of NLP for RE tools is available, Arendse and Lucassen apply three tools of type I on 112 requirements (Arendse and Lucassen, 2016). They find that none of the available tools is clearly superior and call for a next generation tool framework that can dynamically incorporate the new state of the art.

The objective of type II tools, generating models from NL requirements descriptions is a long-standing research topic that is relevant in several domains. Already in 1989, Saeki et al. described a method where verbs and nouns are automatically extracted from NL, in order for a requirements engineer to derive a formal specification of the system (Saeki et al., 1989). One of the first tools that implement this idea is NL-OOPS (Mich, 1996). The authors demonstrate the capabilities of NL-OOPS by generating a data model from a 250 word text. Since then, many tools have been proposed with diverse approaches and results. CM-builder (Harmain and Gaizauskas, 2003) managed to extract *candidate* attributes, entities and relationships from a 220 word text with 73% recall and 66% precision. CIRCE (Ambriola and Gervasi, 2006) is a sophisticated tool that generates many different models including ERD, UML and DFD from NL requirements. Experimental application in three case studies indicated improvements in software model analysis and changing requirements. All these tools, however, require either human intervention or artificially restricted NL in the form of a controlled vocabulary and grammar in order to generate complete and consistent models, thereby hampering adoption in practice. Note that while the Connextra user story template required by our solution imposes a three-part structure, each part is completely free form.

Recognizing this gap in a structured literature review, Yue et al. called for future approaches that fully automatically generate complete, consistent and correct UML models (Yue et al., 2010). Their latest tool, aToucan, generates reasonably high quality class diagrams from use cases in comparison to diagrams created by experts, managing to consistently outperform fourth-year software engineering students in terms of completeness, consistency and redundancy. However, its output constitutes initial models that require human intervention in the form of refinements done by experts (Yue et al.,

2015). Similarly, the tool presented in (Sagar and Abirami, 2014) outperforms novice human modelers in generating conceptual models from natural language requirements. Overall, their recall for identifying entities ranges from 85% to 100% depending on the case, while their precision is between 81% and 94%. The performance for relationships between entities, however, is less impressive: recall is in between 50% and 60%, while precision is in the 80%-100% range. Arora et al. (Arora et al., 2016) take a similar approach to the work we present in this paper, combining existing state-of-the-art heuristics for domain model extraction. They apply their approach to four industrial requirements documents. According to expert evaluation the extraction rules in their implementation achieve correctness between 74% and 100%.

Although (Wautelet et al., 2016a,b; Mesquita et al., 2015) discussed in Sec. 5.6.1 all introduce methods and tools for extracting models from user stories, these works lack empirical evaluations. Because of this, comparing their accuracy with the current state of the art or our work is impossible. Nevertheless, the proliferation of work on extracting models from user stories using NLP signifies the relevance and timeliness of our work. This paper is the first to empirically demonstrate user stories' potential in automatically extracting conceptual models and not without merit. The results of our evaluation show that our approach outperforms aforementioned studies in recall and precision for both identifying concepts as well as relationships. Note, however, that all of the studies employ different evaluation methods and that we have not yet investigated whether Visual Narrator outperforms human requirements engineers.

5.6.3 *Visualization of Conceptual Models and Requirements*

To take advantage of human's visual processing power conceptual models are typically presented as a visualization. Typically, new visual notation initiatives first concern is formally specifying the information types to support. As the popularity of a graphical notation increases, the community around it proposes innovative features to augment or simplify the notation. Unfortunately, aesthetic aspects such as attractive design and user experience are mostly disregarded when it comes to conceptual models (Gulden and Reijers, 2015). Indeed, the most popular conceptual modeling paradigms such as ER, UML and BPMN primarily distinguish elements using basic symbols and shapes (Genon et al., 2010).

There is a large body of work to inspire innovative visualization. For example, in 2009 Moody proposed the Physics of Notations (Moody, 2009)

that provides some guidelines for creating cognitively effective notations and for pinpointing flaws in existing notations such as *i** (Moody et al., 2010) and use case maps (Genon et al., 2010). Moody's guidelines inspired further research that is relevant for our work. For example, Dudáš, Zamazal and Svátek identified seventeen relevant features implemented by ontology visualization tools (Dudáš et al., 2014) such as *incremental exploration* and *fisheye distortion*. Their notation, however, is mostly disregarded. Aside from this, other domains such as interaction design and geography are considerably more sophisticated in effectively visualizing information (Chi, 2000).

A systematic review of the requirements engineering visualization (REV) literature by Abad et al. (Abad et al., 2016) concludes that more investigation and research is needed to support knowledge visualization for RE. Similarly, Cooper et al. (Cooper Jr et al., 2009) review the papers that appeared in the REV workshops between 2006 and 2008. They distinguish between different types of visualizations: tabular, relational, sequential, hierarchical, and metaphorical/quantitative. The most relevant categories for our work are the relational (i.e., graphs) and hierarchical (decomposing a system into its parts). While many relational approaches exist, very few focus on hierarchal aspects, which are the key in our work.

Indeed, there are few contemporary tools available that focus on visualizing requirements as models. To date, ReCVisu+ (Reddivari et al., 2014) is the most effective tool for requirements visualization. ReCVisu+ supports different visual exploration tasks and employs clustering techniques and semantic similarity to reduce complexity. While Visual Narrator only visualizes one type of entity and directed relationship, our work on Interactive Narrator continuously explores innovative techniques to convey additional meaning. For this we take inspiration from existing tools, yet are careful to consider their appropriateness in our context. While RecVisu+ determines similarity based on the frequency of co-occurrence in system documentation, the Interactive Narrator we propose in (Lucassen et al., 2016c) relies on corpus-based techniques that do not require the existence of additional system documentation. Moreover, we do not consider only concepts but also relationships, necessitating a different yet similar approach.

5.7 CONCLUSION AND FUTURE WORK

Natural language is the most adopted notation for requirements (Kassab et al., 2014). Unfortunately, text does not readily provide a holistic view of the involved entities and relationships. Aligned with other authors (Omar et al.,

2004; Du and Metzler, 2006; Harmain and Gaizauskas, 2003; Hartmann and Link, 2007), we argue that extracting a conceptual model can significantly ease the communication between stakeholders.

We have proposed the Grimm method that combines three NLP-enabled tools to support conducting user story based RE. Specifically, the main artifact described in this paper is the Visual Narrator tool, which automatically generates a conceptual model from a collection of agile requirements expressed as user stories. To do so, the tool orchestrates a selection of state-of-the-art NL processing heuristics and relies on an off-the-shelf NLP toolkit, spaCy.

Our evaluation on four case studies showed positive accuracy results, especially when user stories are concise statements of the problem to solve (Cohn, 2004) and not long descriptions of the solution. In comparison to state-of-the-art tools such as (Sagar and Abirami, 2014), our approach performs similarly in terms of entity recall and precision, but is superior in relationship recall. This happens thanks to the careful selection and implementation of heuristics that can deliver accurate results combined with our application of these heuristics based on careful dissection of user stories' syntactical properties.

Improvements can be made to the user story processing algorithms. In particular, we want to develop support for elaborate entities such as complex compounds, n-ary relationships, conjunctions, adjectives, adverbs and references such as 'this' and 'that'. While doing so, however, we aim to make considerate decisions based on maximizing accuracy.

Furthermore, we intend to create a fully functional version of the Interactive Narrator that integrates with the AQUSA Tool and Visual Narrator. We will then empirically evaluate the Grimm method introduced in Section 5.2.2 with practitioners.

Another direction is to study the relationship between user story metrics such as average number of entities, relationships and their density (see Table 5.2) and quality of the created specifications and software. To do so, we need to collect data about how the user stories were used in the later stages of software engineering.

Finally, we want to investigate automated techniques to connect the generated conceptual models to software architecture views. One promising direction is taking advantage of automated acceptance tests, which typically include a hand-coded cross-reference to the user story they test (Cleland-Huang, 2012). For interpreted languages, executing these test cases produces a runtime trace related to the user story. By extracting the accessed data entities, classes and methods it is possible to dynamically construct a class diagram that includes only those classes contained within the runtime trace.

Visualizing User Story Requirements at Multiple Granularity Levels via Semantic Relatedness

6

The majority of practitioners express software requirements using natural text notations such as user stories. Despite the readability of text, it is hard for people to build an accurate mental image of the most relevant entities and relationships. Even converting requirements to conceptual models is not sufficient: as the number of requirements and concepts grows, obtaining a holistic view of the requirements becomes increasingly difficult and, eventually, practically impossible. In this paper, we introduce and experiment with a novel, automated method for visualizing requirements—by showing the concepts the text references and their relationships—at different levels of granularity. We build on two pillars: (i) *clustering techniques* for grouping elements into coherent sets so that a simplified overview of the concepts can be created, and (ii) state-of-the-art, corpus-based *semantic relatedness algorithms* between words to measure the extent to which two concepts are related. We build a proof-of-concept tool and evaluate our approach by applying it to requirements from four real-world data sets.

This work was originally published as:

G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Visualizing user story requirements at multiple granularity levels via semantic relatedness. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, pages 463–478, 2016c

6.1 BACKGROUND

To ease constructing a mental image of a software system, we propose to take requirements expressed as user stories and derive a conceptual model that enables inspecting system functionality with different degrees of granularity. We review the background work that will later be combined as part of our method in Sec. 6.3.

6.1.1 *From User Stories to Conceptual Models*

User stories are a textual language for expressing requirements that uses a compact template. A user story captures *who* the requirement is for, *what* it is expected from the system, and (optionally) *why* it is important (Wautelet et al., 2014). Although many different templates exist, 70% of practitioners use the Connextra template (Lucassen et al., 2016a): *"As a ⟨type of user⟩ , I want ⟨goal⟩, [so that ⟨some reason⟩]"*. For example: *"As an Event Organizer, I want to receive an email when a contact form is submitted, so that I can respond to it"*.

In previous work, we created a tool that automatically extracts a conceptual model from a set of user stories using NLP heuristics: the *Visual Narrator*. This tool is itself built upon a conceptual model of user stories that distinguishes between *role*, *means* and *ends* parts of a user story (Lucassen et al., 2016b). By parsing the user stories with spaCy's part-of-speech tagging (spacy.io, 2016) and applying eleven state-of-the-art heuristics, Visual Narrator creates conceptual models of a user story set with up to 86% recall and 81% precision. Visual Narrator outputs a Prolog program or OWL2 ontology with the concepts and relationships extracted from a user story set.

6.1.2 *Novel Approaches to Semantic Similarity*

As we aim to group concepts in order to facilitate comprehension, we need to identify concepts that are similar or related to one another. We rely on the *semantic similarity* (more precisely, *semantic relatedness*) of word pairs. This is a number—typically in the [0,1] range—that captures the distance between the two words, with 0 being no relatedness and 1 being full relatedness. For any given word, this technique can be used to identify a list of similar words or to calculate its semantic similarity score with a collection of words. If the process is repeated for all words in a collection C, one obtains a matrix that defines the pairwise similarity between all concepts in C.

Among the many approaches to calculating semantic similarity (Harispe et al., 2015), we focus on a family of novel, state-of-the-art algorithms: *skip-gram* by Google (Mikolov et al., 2013) and *GloVe* by Stanford (Pennington et al., 2014). Both algorithms parse huge quantities of unannotated text to generate *word embeddings* without requiring supervision. A word embedding maps some attributes of a word to a *vector* of real numbers that can then be used for a variety of tasks, similarity being one of them. Both skip-gram and GloVe adhere to the distributional hypothesis: *"linguistic items with similar distributions have similar meanings"*.

These techniques constitute the most accurate state-of-the-art and provide significant improvements even on other word embedding approaches (Mikolov et al., 2013; Pennington et al., 2014). Moreover, the innovative vector-based approach of word2vec enables basic "semantic arithmetics" on words: vector("King") - vector("Man") + vector("Woman") results in a vector which is most similar to vector("Queen"). These new methods have not yet been applied in the conceptual modeling and RE literature, while they are slowly but steadily being adopted in industry, also thanks to their excellent performance.

6.1.3 *Clustering Algorithms*

Clustering refers to the process of taking a set of concepts and grouping them so that concepts within the same group are similar and concepts in other groups are different. We aim to adopt clustering in the context of user stories on the basis of the semantic similarity/relatedness between concepts. Since word embeddings and semantic similarity scores are expressed as numbers, we can easily use existing tools to apply the existing variety of clustering algorithms.

The go-to algorithm for most clustering needs is *k*-means , but through experimentation with many of the available algorithms we found it to be less applicable for our use case. This is mostly due to the randomness of the algorithm: the resulting clusters differentiate too much between runs.

Instead, we choose an algorithm that leads to similar accuracy results but uses a consistent approach: Ward's minimum variance method (Ward, 1963). Ward's algorithm starts by assigning all concepts to their own cluster and then iterating over the cluster collection until it finds the two clusters that, when merged, lead to a minimum increase in within-cluster variance of the collection of clusters. It keeps repeating this step until k clusters have been formed. Although Ward's method is slower than k-means, the impact is negligible for the relatively small data sets that one extracts from a user story set.

6.2 INTRODUCTION

Natural language (NL) is the most popular notation to represent software requirements: around 60% of practitioners employ NL as their main artifact (Kassab et al., 2014). Moreover, the trend in agile development has boosted the adoption of the semi-structured NL notation of *user stories* (Wang et al., 2014; Kassab, 2015; Lucassen et al., 2016a): "As a ⟨*role*⟩, I want ⟨*goal*⟩, so that ⟨*benefit*⟩". Recent research (Lucassen et al., 2016a) shows that 90% of practitioners in agile development adopt user stories.

NL requirements are easy to read but have a major drawback: as their number increases, the quantity of the involved concepts grows rapidly, making it increasingly harder for humans to construct an accurate mental model of those concepts. A possible solution is the (semi-)automated generation of an explicit conceptual model (Omar et al., 2004; Du and Metzler, 2006; Harmain and Gaizauskas, 2003).

Inspired by these works, we have previously proposed the *Visual Narrator* tool that automatically extracts conceptual models from sets of user stories with satisfactory accuracy (80%-90%) (Robeer et al., 2016). However, our evaluation with practitioners indicated that the extracted models quickly become too large to be effectively explored by analysts.

The problem of model comprehensibility can be generalized to the conceptual modeling field (Aranda et al., 2007; Moody, 2009): humans' working-memory capacity restricts the ability to read models and leads to *cognitive overload* when the same model includes too many concepts.

To tackle this problem, we build upon Shneiderman's *visual information seeking* mantra: *"overview first, zoom/filter, details on demand"* (Shneiderman, 1996). We propose and experiment with a mechanism for improving the visualization of conceptual models that are generated by the *Visual Narrator* from user stories. We make use of clustering techniques to group the concepts so to obtain an initial *overview*.

We go beyond existing clustering approaches in literature (see Sec. 6.5) by leveraging on state-of-the-art, corpus-based *semantic relatedness* algorithms based on neural networks to determine the similarity between concepts (Goldberg and Levy, 2014; Trask et al., 2015). The significant improvements in accuracy of these recent approaches trigger our experimentation of these techniques for guiding the clustering of the concepts in user stories.

Our approach is novel in that it does not require additional documentation about the system under development; indeed, it relies on publicly available corpuses of data from the Web. Moreover, we focus on conceptual models

generated from user stories, which have a limited expressiveness compared to full-fledged conceptual models.

Specifically, this paper makes two contributions:

- We devise a method for creating an overview of the concepts in a conceptual model by creating clusters that contain semantically related concepts;
- We propose a proof-of-concept tool for generating clusters and for zooming them; we evaluate its feasibility by applying it to user story sets from 4 real-world cases.

The rest of the paper is structured as follows. Sec 6.1 outlines our background: the *Visual Narrator*, semantic similarity, and clustering techniques. Sec. 6.3 presents our method. Sec 6.4 applies our proof-of-concept tool to real-world user story data sets. Sec. 6.5 reviews related work, and Sec. 6.6 presents our conclusions and future directions.

6.3 VISUALIZATION METHOD FOR USER STORIES

We describe our approach to visualizing concepts and relationships between user stories based on the theory introduced in Sec. 6.1. Our method features three main functionalities: the generation of an overview (Sec. 6.3.1), zooming in and out mechanisms (Sec. 6.3.2), and filtering techniques (Sec. 6.3.3). To illustrate, we use a publicly available set of 104 user stories from the Neurohub project, an information environment for neuro-scientists developed by three British universities.

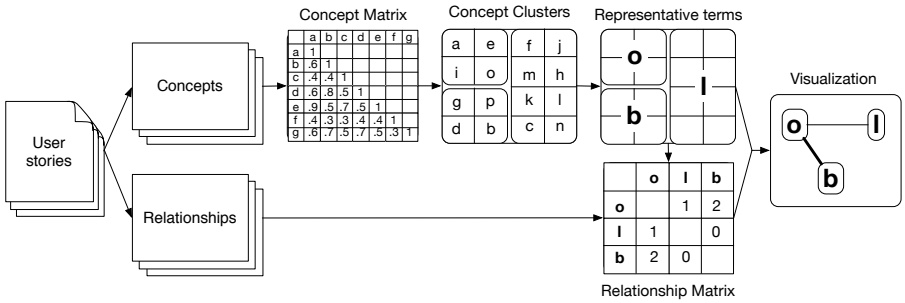


Figure 6.1: Our method for generating the overview

6.3.1 *Overview generation*

The purpose of an overview is to *"provide a general context for understanding the data set"* (Craft and Cairns, 2005). By abstracting from all the details of the data, filtering extraneous information and highlighting specific patterns and themes in the data, the overview supports the end-user in understanding the information. To achieve this goal for a user story set, we propose a 6-step process visualized in Fig. 6.1 and elaborated below.

1. **Extraction from User Stories.** The Visual Narrator extracts a set of relevant concepts C from the user stories and a set of relationships $R \subseteq C \times C$ between those concepts. In our example, the output consists of 124 concepts in C and 144 relationships between these concepts in R as shown in the following Prolog lines:

   ```
   concept('Neuroscience').
   concept('Researcher').
   ...
   isa(concept('Book Page'),concept('Page')).
   rel(concept('Researcher'),'Create',concept('Book Page')).
   ```

2. **Concept Similarity Calculation.** We use the skip-gram implementation *word2vec*[1] to calculate the semantic similarity/relatedness scores—in the range [0,1]—for each concept with all other concepts in the list C. As explained in Sec. 6.1, the use of skip-grams combines efficiency and accuracy. This step results in a similarity matrix SM of size $|C| \times |C|$ such that $\forall i, j \in [0, |C|)$. $SM_{i,j} = skipgram(c_i, c_j)$. In the following example, it is possible to see how some couples of words are much more semantically related than others: compare researcher and neuroscience (0.5134) with neuroscience and booking (0.1667).

SM	neuroscience	researcher	book	booking	...
neuroscience	1.0000	0.5134	0.3446	0.1667	...
researcher	0.5134	1.0000	0.3362	0.2055	...
book	0.3446	0.3362	1.0000	0.2301	...
booking	0.1667	0.2055	0.2301	1.0000	...
...	...	...	...	...	...

[1] https://code.google.com/p/word2vec/

3. **Concept Clustering.** We utilize Ward's clustering algorithm to group all the concepts according to their similarity in SM. This results in a high-level disjoint clustering WC that forms the basis for our visualization. In our experimentation, inspired by the cognitive principles by Moody (Moody and Flitman, 1999), we generate nine clusters.

```
0: ['acceptance test', 'acceptance', 'analysis', 'behaviour', 'dependency', 'description', '
    drug response', 'experiment description', 'experiment', 'input', 'knowledge', '
    neurohub dependency', 'neuroscience', 'paper', 'provenance', 'research paper', '
    research', 'response', 'search result', 'search', 'southampton neuroscience', 'test
    result', 'test', 'work', 'worm analysis']
1: ['control system', 'equipment booking', 'equipment', 'installation', 'lts machine', 'lts', '
    lab administrator', 'lab member', 'lab', 'mri operator', 'mri', 'neurohub installation'
    , 'neurohub workspace', 'spreadsheet', 'system administrator', 'system', 'workspace']
2: ['behaviour video', 'calendar', 'directory', 'google calendar', 'google', 'inventory', 'link',
     'log', 'machine', 'meta', 'reference', 'script', 'table', 'tag', 'type', 'ups', 'version', '
    video', 'web', 'worm', 'write ups', 'write']
3: ['browser', 'client', 'interface graphics/colour', 'interface', 'mendeley client', 'neurohub
    node', 'node', 'operator', 'protocol', 'user', 'web browser']
4: ['booking', 'control', 'cost', 'drug', 'event', 'field', 'forward', 'minimal', 'others', 'period'
    , 'release', 'result', 'run', 'share', 'sharing', 'southampton', 'time', 'track', 'what']
5: ['data file', 'data', 'file type', 'file', 'html tag', 'html', 'information', 'keywords', 'meta
    data', 'metadata', 'minimal information', 'share data', 'template']
6: ['administrator', 'engineer', 'investigator', 'member', 'release engineer', 'researcher', '
    supervisor']
7: ['graphics/colour', 'mendeley', 'neurohub']
8: ['book entry', 'book page', 'book', 'entry', 'log book', 'neurohub page', 'page']
```

Note the clusters' different sizes: while cluster 0 has size 25, cluster 7 has size 3. Also, cluster 6 neatly relates roles/professions such as administrator, engineer, etc. This is one of the key differences of employing corpus-based similarity as opposed to looking at the graph structure.

4. **Representative Term Selection.** From each cluster c_i in WC, we identify the concept which is most similar to the collection of concepts in cluster c_i. We do so by using the analogy capabilities of skip-gram as introduced in Section 6.1.2: we compute the sum of the word vectors swv of the concepts in a cluster; then, we set the cluster label by choosing the name of concept in the cluster whose vector model is most similar to the vector swv. For example, consider a cluster with concepts administrator, visitor and individual. We compute the sum vector swv = vector("Administrator") + vector("Visitor") + vector("Individual"). Among these concepts, the word whose vector is closest to the sum of the vectors is individual. To avoid meaningless labels, we remove stop words—such as he, a, from, ...—before we execute this step. For the Neurohub case, we obtain the following results:

0: analysis	1: lab	2: web	3: user	4: time
5: data	6: engineer	7:	8: book	

Note that for cluster 7 no label could be assigned because word2vec does not have any of the cluster's terms in its dictionary.

5. **Inter-cluster Relationships Matrix Generation.** Since the concepts in a cluster are represented by one term, intra-cluster relationships do not need to be visualized. Starting from the list of relationships R, we derive a matrix ICR of size $|WC| \times |WC|$ that determines the strength of the relationships between the clusters by counting how many relationships exist between the concepts in those clusters. Formally, $\forall i, j \in [0, |WC|)$:

$$ICR(c_i, c_j) = \begin{cases} 0, if\ i = j \\ |r(x,y)| \in R.\ (x \in c_i \wedge y \in c_j) \vee (x \in c_j \wedge y \in c_x), else \end{cases}$$

In our example, we obtain the following matrix:

ICR	$c0$	$c1$	$c2$	$c3$	$c4$	$c5$	$c6$	$c7$	$c8$
$c0$	0	2	2	4	5	0	3	1	0
$c1$	2	0	2	3	3	1	5	2	0
$c2$	2	2	0	8	0	4	4	0	1
$c3$	4	3	8	0	7	10	1	3	2
$c4$	5	3	0	7	0	2	5	0	0
$c5$	0	1	4	10	2	0	14	0	0
$c6$	3	5	4	1	5	14	0	0	1
$c7$	1	2	0	3	0	0	0	0	1
$c8$	0	0	1	2	0	0	1	1	0

The remarkably large number of relationships between 5 and 6 is caused by the concentration of role concepts in cluster 6 who perform an action on the concepts related to data in cluster 5.

6. **Visualization Drawing.** Each cluster $c \in WC$ becomes a vertex (a circle) with the representative term as its label. The diameter of the circle increases with the number of concepts in the cluster. Lines are drawn for each inter-cluster relationship in ICR; the width of a link increases with the number of relationships between the connected clusters. An example of the generated overview is shown in Fig. 6.2.

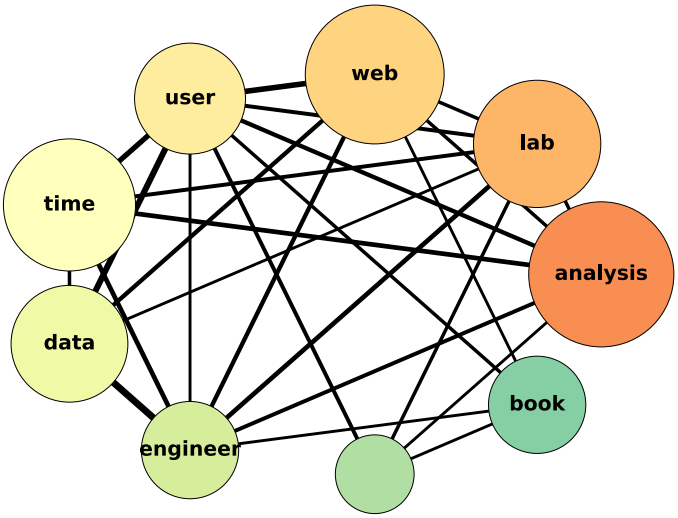


Figure 6.2: Example overview of the user stories from the Neurohub project

6.3.2 *Zooming*

The purpose of zooming is to reduce the complexity of the data presentation by having the user adjust the data element size and selection on the screen (Craft and Cairns, 2005). We propose two zooming views that enable exploring distinct details of the overview; these views are accessed by clicking on either (1) a concept or (2) a relationship.

When a user clicks on a concept, that concept will be zoomed in and the steps outlined in Sec. 6.3.1 are re-run on the concepts within the cluster. The only difference is that we set the number of sub-clusters to the square root of the number of elements within the cluster[2]. The outcome is a more granular view of the concepts in the cluster and showing new inter-cluster relationships that were previously hidden as intra-cluster relationships. See the left image of Fig. 6.3 for an example. In case the number of concepts in that cluster is lower or equal than 9 (see (Moody and Flitman, 1999)), all concepts are shown.

Clicking on a relationship will simultaneously zoom in on the two clusters that the relationship connects, showing the same more granular view of the concepts as when clicking on a concept. Furthermore, zooming in on the relationship displays all underlying relationships individually and connects the smaller underlying concept clusters on both sides. See the right image of Fig. 6.3 for an example.

[2]Determining the number of clusters is still a work-in-progress part of our approach

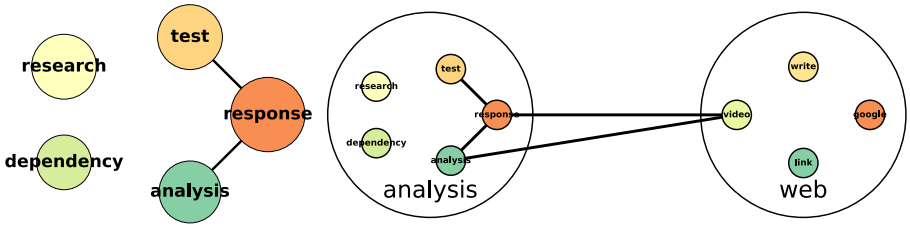


Figure 6.3: Zooming examples for Neurohub: concept zoom on the *analysis* cluster (on the left), and relationship zoom between the *analysis* and the *web* clusters (on the right)

6.3.3 *Filtering*

The purpose of filtering is the same as zooming: reducing the complexity of the presented data. However, instead of selecting a specific region, filtering controls enable the user to change whether the data points with a given attribute are visible (Craft and Cairns, 2005). In the context of user stories, we propose four filters that enable the user to further simplify a data view or to further explore some specific details:

1. *Relationships:* by default any view is drawn with its relevant relationships. Optionally, the user can turn this off the relationships clicking, allowing complete focus on the concept clusters. Alternatively, it is possible to filter out specific relationship types, e.g., visualizing only or hiding is-a relationships.
2. *Role:* a central and prominent aspect of user stories, roles are the most frequently occurring concepts in any user story set. Indeed, 96 of the 123 relationships in the Neurohub example (78%) connect a role to some concept. We propose two ways of filtering roles: (i) removing all roles from the set of concepts C, enabling the user to focus on relationships between other concepts; (2) selecting a specific role to focus on, removing all concepts and relationships that are not related to that role.
3. *Search:* users can query for concept terms to find the cluster related to that concept. For example, searching for *file* will highlight the data cluster and its relationships while slightly blurring all unrelated concepts and relationships.
4. *Agile Artifacts:* In agile software development, user stories are organized into meaningful chunks: epics, themes and sprints. The user can select any combination of these in order to explore specific parts of the system (via epics and themes) or to focus on certain development periods (sprints).

6.4 PROTOTYPE DEMONSTRATION

We demonstrate the feasibility of our approach by applying a prototype implementation to three real-world case studies. The prototype is available online including the Neurohub user stories[3]. Unfortunately, we cannot release the confidential case study user stories. For each case, we present four views of their user story concepts and relationships: overview, concept zoom, relationship zoom and a role filter. Finally, we evaluate and discuss the output.

6.4.1 *Case 1: CMSCompany*

The company developing this complex CMS product for large enterprises is located in the Netherlands, serving $\tilde{1}$50 customers with 120 employees. Their supplied data consists of 32 syntactically correct user stories, which represents a snapshot of approximately a year of development in 2011. Visual Narrator extracted 96 concepts and 114 relationships, exemplifying that despite the small size of the user story set, the use of long user stories with non-trivial sentence structuring means many concepts and relationships are present.

Applying the prototype to *CMSCompany*'s user stories results in Fig. 6.4. Upon examination of the overview on the left, one thing immediately stands out: the clusters are highly interrelated and none of them clearly has the majority of relationships. This is likely a consequence of the long, non-trivial structuring of these user stories. Furthermore, some of the representative terms are highly relevant to the CMS domain: site, marketeer, text, business & analytics are important aspects of CMSCompany's product.

By contrast, the concept zoom of the *result* cluster has no intercluster relationships at all. In fact, none of the subclusters contain such a relationship. Intuitively, the authors believed this to be a bad result but upon closer examination this phenomenon actually turns out to be the ideal situation. Indeed, there *are* relationships between concepts in this subcluster but they are all within their own subsubcluster, demonstrating that at this level the semantic clustering approach is very effective at grouping related user story concepts.

[3]https://github.com/gglucass/Semantic-Similarity-Prototype

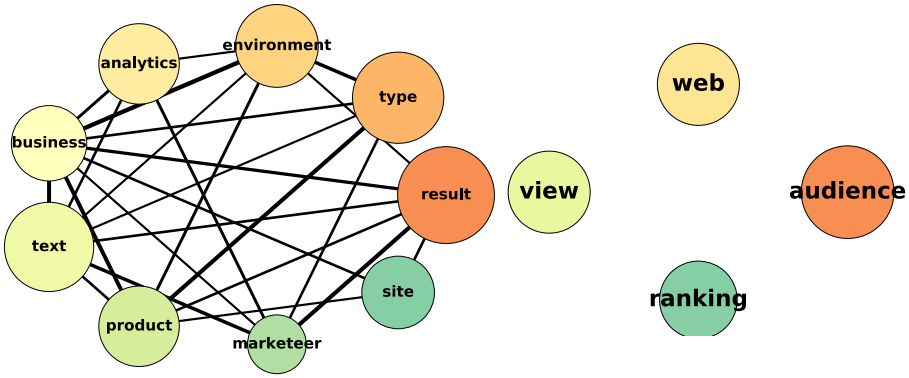


Figure 6.4: Overview for CMSCompany and concept zoom on *result* cluster

6.4.2 *Case 2: WebCompany*

This is a young Dutch company that creates tailor-made web applications for businesses. The team consists of nine employees who iteratively develop applications in weekly Scrum sprints. WebCompany supplied 98 user stories covering the development of an entire web application focused on interactive story telling that was created in 2014. Although the data set is 3x as big as CMSCompany's, these user stories are very simple, concise and contain very few complex sentence structures. Because of this, Visual Narrator extracts just 106 concepts and 123 relationships.

In CMSCompany's overview in Fig. 6.5 the *person* cluster clearly has the most relationships with other clusters. This is a direct consequence of two factors: (1) the person clusters contains all roles defined in the user stories and (2) because the user stories are simple, the majority of relationships in this set are role(action, object). However, for this case not all representative

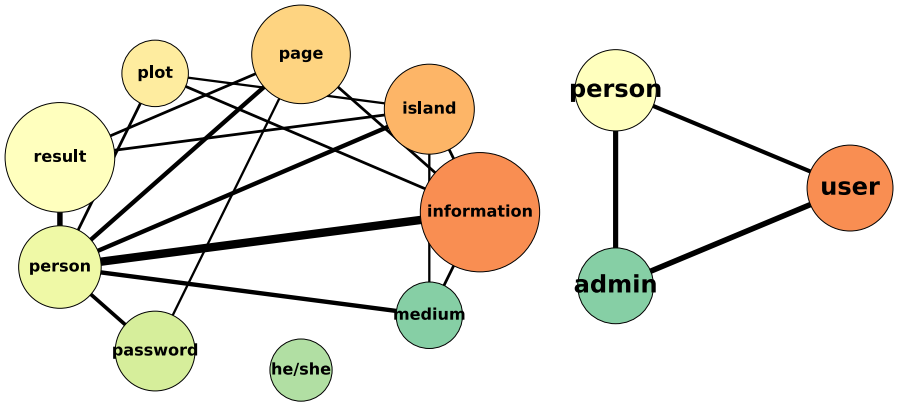


Figure 6.5: Overview for WebCompany and concept zoom of *person* cluster

terms are meaningful. In particular, *result* is strongly related to merely 3 or 4 out of 22 concepts in that cluster. This exemplifies the approach's weakness of selecting very general terms for large, less coherent clusters because they are at least somewhat related to many of the terms in the cluster.

In some cases, previously intracluster relationships do become intercluster relationships when zooming in on a cluster. The subclusters in *person* are all related to one another in some way. Because this user story concerns an entire web application, this is to be expected. Indeed, if admin was not related to a user, a human analyst should be triggered to investigate if the user story set is incomplete. This exemplifies a possible real-world use case of the prototype.

6.4.3 *Case 3: SCMCompany*

This case concerns stories from a company that delivers a leading Supply Chain Management (SCM) suite for large companies in the logistics, healthcare, technology and consumer sectors. To support development in keeping up with double digit revenue growth, the company has started to embrace user stories. This set consists of 54 high quality user stories of moderate size and complexity. In total, Visual Narrator extracted 91 concepts and 114 relationships.

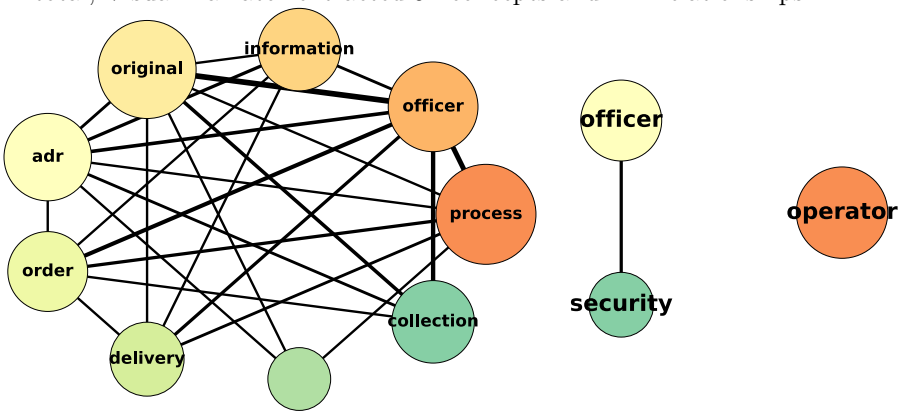


Figure 6.6: Overview for SCMCompany and concept zoom of *officer* cluster

Nearly all of the representative terms in the overview in Fig. 6.6 are strongly related to the SCM domain. Furthermore, the relationship between concepts *security* and *officer* in the concept zoom is actually an accurate representation of the relationship in the source data. Considering that the sub clusters contain multiple concepts, this example demonstrates a positive result of our approach for selecting a representative term.

6.5 RELATED LITERATURE

We review the relevant literature about RE visualization, clustering techniques for generic conceptual models, and extraction of conceptual models from requirements.

6.5.1 *RE visualization*

Requirements engineering visualization (REV) is concerned with creating effective visualizations of RE artifacts. Cooper et al. (Cooper Jr et al., 2009) review the papers appeared in the REV workshops between 2006 and 2008. They distinguish between different types of visualizations: tabular, relational, sequential, hierarchical, and metaphorical/quantitative. The most relevant categories for our work are the relational—i.e., (hyper-)graphs—and hierarchical—decomposing a system into its parts—. While many relational approaches exist, very few focus on hierarchal aspects, which are the key in our work.

We deliberately exclude from this section the vast body of literature on requirements modeling languages: this important family of requirements visualization approaches goes beyond the scope of our paper, as we aim at visualizing the main concepts that can be extracted from NL requirements.

Reinhard et al. (Reinhard et al., 2007) propose an improved Fisheye zoom algorithm for the visualization and editing of hierarchical requirements models. The most interesting feature of their algorithm is that is guarantees stability of the adjusted layouts and runs in linear time. We may exploit this algorithm in future work. Gandhi and Lee (Gandhi and Lee, 2007) use visualizations in the context of requirements-driven risk assessment. They extract a concept lattice that relates risk-related concepts such as assets, threats, vulnerabilities and countermeasures. The lattice is a possible criterion for a zoom-in/zoom-out mechanism.

To date, ReCVisu+ (Reddivari et al., 2014) is the most effective tool for requirements visualization. ReCVisu+ supports different visual exploration tasks and, like our approach, is based on clustering techniques and semantic similarity. While RecVisu+ determines similarity based on the frequency of co-occurrence in system documentation, we rely on corpus-based techniques that do not require the existence of additional system documentation. Moreover, we do not consider only concepts but also relationships.

6.5.2 *Conceptual modeling clustering*

The conceptual modeling and databases community is well aware that (E)ER diagrams are often large and cluttered. Teorey et al. have studied this problem already in the late 80s (Teorey et al., 1989): they proposed collapse/expand mechanisms that can be used to hide/view entities and relationships that are secondary to some primary entities; for example, 'journal address' and 'journal note' can be collapsed into a cluster labeled 'journal'.

Akoka and Comyn-Wattiau (Akoka and Comyn-Wattiau, 1996) propose automated clustering techniques that can be used to realize Teorey's vision and that derives non-overlapping clusters. They experiment multiple distance indicators with different semantics (visual, hierarchical, cohesive, etc.) and compare their strengths and weaknesses. Moody and Flitman (Moody and Flitman, 1999) combine and refine principles from previous work and include cognitive aspects, such as the maximum size of a cluster being nine elements, in order to facilitate human comprehension. Tzitzikas and Hainaut (Tzitzikas and Hainaut, 2005) use link analysis from web searching to generate a smaller diagram that includes only the major entity and relationships; they also propose an automated 2D visualization that uses force-directed drawing algorithms.

Summarization techniques exist for database schemas. Yu and Jagadish (Yu and Jagadish, 2006) formalize the notion of a schema summary and define its quality in terms of complexity (number of elements), importance (inclusion of the most important elements), and coverage (are all major chunks of the original schema represented?). Based on these notions, they propose algorithms that optimize each of these quality criteria. Yuan et al. (Yuan et al., 2014) go further by proposing elaborate metrics to determine table similarity and importance.

All these techniques inspire our work. The main novelty of our proposal is that we focus on conceptual models that represent requirements, and that we use novel corpus-based techniques to determine similarity. The significant advances that these algorithms provide make it possible for us to experiment clustering based on the co-occurrence of words in corpora of data on the Web with promising results.

6.5.3 *Extracting conceptual models from requirements*

There is a long tradition in generating conceptual models from NL requirements. Already in 1989, Saeki et al. (Saeki et al., 1989) proposed a method for the automatic extraction of verbs and nouns from NL. The ideas of this method were operationalized by numerous (semi-)automated tools, includ-

ing NL-OOPS (Mich, 1996), CM-Builder (Harmain and Gaizauskas, 2003), CIRCE (Ambriola and Gervasi, 2006), aToucan (Yue et al., 2015), and the tool by Sagar and Abirami (Sagar and Abirami, 2014).

All these approaches use NL processing algorithms such as tokenization, part-of-speech tagging, morphological analysis and parsing. These tools show promising precision and recall—comparable if not better than human experts—, although they often require restricted NL to do so. In previous work (Robeer et al., 2016), we leveraged these tools and proposed an approach that is specifically suited for requirements expressed as user stories.

6.6 DISCUSSION, CONCLUSION AND FUTURE WORK

This paper explored the potential of applying semantic relatedness algorithms for visualizing user stories. After studying and experimenting with state-of-the-art algorithms such as skip-gram, we presented a novel, automated method for visualizing user stories at different levels of granularity. We applied a prototype implementation of this method to four real-world user story sets, studied the output and observed that:

- The generated visualizations are capable of highlighting relevant information classifications for the system. For example, the central role of the person cluster in the WebCompany overview is easily recognizable.
- For the majority of clusters, the generated representative term is meaningful and relevant within the application domain.
- When an intercluster relationship is present on the zoom level, it is generally relevant within that (sub-)domain. The analysis subcluster of Neurohub for example relates test, response and analysis.
- On the overview level, too many intercluster relationships are visible, effectively rendering them useless for further human analysis.
- The prototype tends to select irrelevant common denominator terms for large clusters with low internal coherence.
- Word2Vec does not include all words in meaningful clusters, resulting in residual clusters that cannot be assigned any label.

Based on these observations, we envision possible applications for our visualization approach to include: (1) discovering missing relationships between clusters that may result in further user stories; (2) teaching system functionality by exploring simplified, manageable chunks; and (3) analyzing expected system changes after introducing new sets of user stories (e.g., new epics).

However, further practitioner evaluation is necessary to confirm the validity of our observations and the potential of these applications.

In future work, we intend to experiment with these possible use cases as well as investigate how to substantially improve the generated output. A necessary next step is to combine and compare our work with existing state-of-the-art clustering techniques that do not rely on semantic relatedness. Additionally, future work should incorporate state-of-the-art group structure visualization techniques (Vehlow et al., 2015). We expect this to produce outputs that are even more usable in real-world scenarios. Additionally, we are investigating the potential benefits of applying machine learning to enhance the accuracy of semantic relatedness scores for specific application domains.

Behavior Driven Requirements Traceability via Automated Acceptance Tests

7

Although information retrieval advances significantly improved automated traceability tools, their accuracy is still far from 100% and therefore they still need human intervention. Furthermore, despite the demonstrated benefits of traceability, many practitioners find the overhead for its creation and maintenance too high. We propose the Behavior Driven Traceability Method (BDT) that takes a different standpoint on automated traceability: we establish ubiquitous traceability between user story requirements and source code by taking advantage of the automated acceptance tests that are created as part of the Behavior Driven Development process.

This work is an extension of:

G. Lucassen, F. Dalpiaz, J. M. E. M. v. d. Werf, S. Brinkkemper, and D. Zowghi. Behavior-driven requirements traceability via automated acceptance tests. In *Proceedings of the International Workshop on Just-In-Time Requirements Engineering (JIT-RE)*, pages 431–434, 2017a

7.1 INTRODUCTION

Software traceability is a long-standing research topic widely regarded as important to creating and maintaining high-quality software (Arkley and Riddle, 2005; Blaauboer et al., 2007; Mäder and Egyed, 2015; Berry et al., 2016). The Center of Excellence for Software and Systems Traceability (CoEST) defines traceability as "the ability to interrelate any uniquely identifiable software engineering artifact to any other, maintain required links over time, and use the resulting network to answer questions of both the software product and its development process" (Cleland-Huang et al., 2014a).

The benefits of maintaining requirements traceability include support for change impact analysis, increased program comprehension and faster software development (Mäder and Egyed, 2015). Nonetheless, many industry practitioners do not adopt traceability practices (Blaauboer et al., 2007). Two main reasons are that (i) the stakeholders who need to create the links are not the ones who reap the benefits, the so-called *traceability benefit problem* (Arkley and Riddle, 2005; Berry et al., 2016), and (ii) the lack of methods and tools supporting traceability (Bouillon et al., 2013) hamper the effective adoption of traceability practices (Cleland-Huang et al., 2014a).

Recognizing these problems, Cleland-Huang and colleagues put forward the grand challenge of "always there" *ubiquitous* traceability that is "built into the software engineering process" (Cleland-Huang et al., 2014a). This is particularly relevant for the ad-hoc and just-in-time requirements engineering practices of agile and open source projects, which do not go beyond prefacing a commit message with an issue ID (Ernst and Murphy, 2012). Many tools have been proposed (Borg et al., 2014; Duarte et al., 2016; Vale et al., 2016) that largely rely on information retrieval (IR) algorithms. Although promising, those IR algorithms do not yield the 100% accuracy that ubiquitous traceability requires and their performance is highly dependent on the input data quality and type (Merten et al., 2016).

In this paper, we present the automated Behavior-Driven Traceability method to establish ubiquitous traceability by taking advantage of *automated acceptance tests*. Created as part of the software engineering process *Behavior-Driven Development* or BDD(North, 2006), these tests refine user story requirements into steps that mirror a user's interaction with the system. A typical acceptance test has at least three steps: (i) establishing a starting position by navigating to a specific interface, (ii) executing one user action such as clicking a button and (iii) asserting whether the action produced the expected effect.

Unlike unit tests, these steps do not directly execute the source code itself. Instead, a BDD framework launches a complete instantiation of the software system and then simulates how a user would interact with the interface. This creates an opportunity to leverage runtime tracers to identify all the source code invoked to realize a given user story without requiring imprecise information retrieval techniques.

After presenting the necessary background (Section 7.2), the rest of the paper makes three concrete contributions towards bridging this gap with ubiquitous traceability:

1. We introduce the *Behavior Driven Traceability* method (BDT) for generating execution traces that link user stories to code via automated acceptance tests (Section 7.3). We illustrate our work with the running example of the fictitious EventCompany's software that enables event organizers to sell tickets to their visitors online.

2. We build on BDT and present the BDT Change Impact Analysis (CIA) method that, using an entity extraction tool (Robeer et al., 2016), automatically generates change impact analysis reports for newly introduced requirements (Section 7.4).

3. We apply prototype tools for both BDT and CIA to the *Archive of Our Own* open source project (1 million users, 100k lines of code, 75 user stories, 710 BDD tests, over 10k test steps) in Section 7.5. We discuss their effectiveness in identifying relevant classes and methods by seeking answers to three evaluation questions:

   Q1. How accurate and relevant are the generated traces?
   Q2. Does trace normalization improve their relevance?
   Q3. Do change impact analysis reports adequately reflect the consequences of new requirements?

We finalize this paper in Section 7.6 by discussing our results, examining the threats to validity and presenting opportunities for future work.

7.2 BACKGROUND

We present the relevant literature for our approach: agile requirements via user stories, requirements traceability and its role in agile development, and behavior driven development.

7.2.1 *User Stories*

User stories are a concise notation for capturing requirements whose adoption has grown to 50% thanks to the increasing popularity of agile development practices such as Scrum (Cao and Ramesh, 2008; Kassab, 2015; Wang et al., 2014). A user story consists of three basic components: (1) a short text describing the user story itself, (2) conversations between stakeholders to exchange perspectives on the user story, and (3) acceptance criteria. In this paper, we are concerned with the first and the third elements.

The first component captures the essential elements of a requirement: *who* it is for, *what* is expected from the system, and, optionally, *why* it is important. The de-facto standard for formulating a user story is the Connextra format (Lucassen et al., 2016a), popularized by Mike Cohn (Cohn, 2004): *"As a ⟨type of user⟩ , I want ⟨goal⟩, [so that ⟨some reason⟩]"*. For example: *"As an Administrator, I want to receive an email when a contact form is submitted, so that I can respond to it"*.

7.2.2 *Requirements traceability*

Requirements traceability has been studied for almost three decades (Cleland-Huang, 2012) and already in 2003 Lee et al. emphasized the importance of early and non-obtrusive methods for traceability to become successful in agile software development (Lee et al., 2003). This is echoed by the vision for *ubiquitous traceability* that is "always there" and "built into the engineering process" formulated by the CoEST and further clarified by Cleland-Huang et al. (Cleland-Huang et al., 2014a): a new developer that is assigned a user story has access to detailed change impact analysis reports that are dynamically generated based on the new user story's context and 100% automatically constructed trace links thanks to tools that take advantage of software development process artifacts.

Recent works contribute towards this vision with varying levels of success. The lightweight just-in-time traceability approach by Cleland-Huang, Rahimi and Mäder separates trusted trace links from untrusted ones to systematically construct trace links that support specific tasks when needed (2014b).

Recognizing that many of the existing requirement-to-code traceability approaches employ just one of three common IR techniques (synonyms, verb-object phrases, and structural information), Zhang et al. present the R2C tool that combines all three. Evaluating four real projects, they find that their tool outperforms the previous state-of-the-art (Zhang et al., 2016).

Rempel, Mäder and Kuschke propose using a graph clustering algorithm to support the retrieval of so-called refinement traces: traces between similar artifacts created in different development phases such as high-level epics and detailed user stories. The results of their experimentation on three datasets indicate graph clustering does not yet achieve ubiquitous traceability but is a step forward towards its goals (Rempel et al., 2013).

Most recently, Rahimi, Goss and Cleland-Huang presented the Trace Link Evolver for automating the evolution of requirements-to-code trace links (Rahimi et al., 2016). Whenever source code is extended or changed, a collection of heuristics, open source tools and IR methods are triggered to detect common change scenarios, associate these scenarios with link evolution heuristics and adjust the trace links accordingly. Their sophisticated approach significantly outperforms the standard IR approaches Latent Semantic Indexing and Vector Space Model.

Unfortunately, industry has yet to embrace advances in ubiquitous traceability. Instead of relying on automatic algorithms, the only agile traceability practices with some industry adoption is *as-needed* traceability. One study found that contributors to three large open source projects preface a commit message to the source code repository with an issue ID (Ernst and Murphy, 2012) to manually create a trace between an issue and the relevant code segment. However, unlike the method we describe in this paper, annotating commits is only exhaustive when applied from the inception of a project, is vulnerable to human negligence throughout the project and requires processing large amounts of historical data.

7.2.3 *Behavior Driven Development*

The origins of Behavior Driven Development (BDD) trace back to a 2006 article by Dan North (North, 2006), consisting of statements to augment Test Driven Development (TDD) (Beck, 2003). The central ideas are that (a) teams should formulate a simple "ubiquitous language" for capturing automated acceptance tests that any team member can read and (b) these acceptance tests should specify user *behavior* for the system to fulfill. North suggests adopting a format that incorporates at least: (i) a title, (ii) a user story in the Connextra format, and (iii) one or more scenarios that test the user story's intended behavior.

Listing 1 Example acceptance tests for EventCompany US1

```
1:  Feature: Contact Form
2:      As a Visitor
3:      I want to use the contact form
4:      so that I can contact the organizer.

5:  Scenario: Submit contact form
6:      Given I go to the "contact" page
7:       visit path_to(contact_page)
8:      When I enter "Hello World" in the "Question" field
9:       fill_in('Question', with: 'Hello World')
10:     And I submit the form
11:      click_button('Submit')
12:     Then the organizer receives a message
13:      open_email('organizer@webcompany.com')
14:      expect(current_email).to have_content 'Hello World'
```

See Listing 1 for an example acceptance test utilizing the industry-standard Gherkin syntax, which describes a series of steps using **given** *some initial context*, **when** *an event occurs*, **then** *ensure some outcome* (Wynne and Hellesoy, 2012). Each scenario step captures a single *behavior* that the system should fulfill. While not ubiquitous, it is recommended practice to annotate a BDD test with the user story it tests (North, 2006). For illustration purposes, we refer to requirements for the EventCompany scenario (see Section 7.1).

Each scenario step captures a single *behavior* for the system to fulfill. If these tests are maintained and extended to support new functionality, a comprehensive BDD test collection functions as a growing, accurate and up-to-date documentation source for the entire system (Wynne and Hellesoy, 2012).

Unlike unit tests, BDD tests do not directly execute methods or code parts. Instead, a BDD test framework launches the entire software system and simulates how a real user would interact with the software's interface step by step. As a consequence, the execution trace of a BDD test includes all the source code invoked to realize its annotated user story. This creates an opportunity to trace user story requirements without requiring probabilistic—thus, imprecise—IR techniques. While the tracing is still only as good as the human-created BDD tests, unlike other approaches BDD output does not require any additional effort to establish ubiquitous traceability.

We illustrate the interaction between automated acceptance tests, BDD frameworks and software on the Contact Form feature of Listing 1 in the sequence diagram of Fig. 7.1.

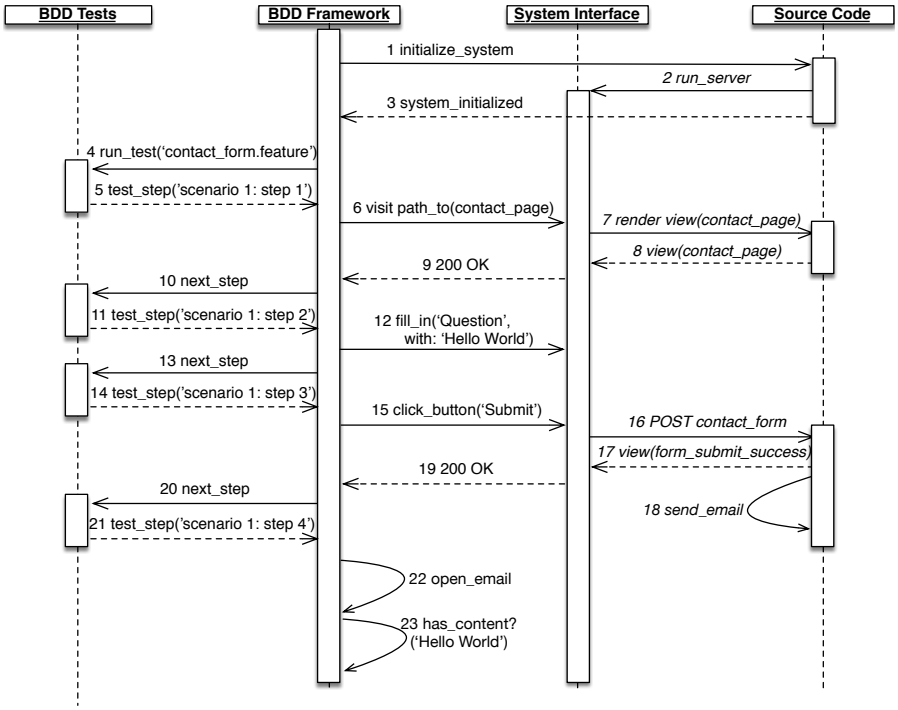


Figure 7.1: BDD Sequence Diagram for Contact Form Feature

The sequence starts when an EventCompany developer decides to run the contact form test with `cucumber contact_form.feature`. This invokes the BDD framework *Cucumber* which first launches an instantiation of the system by initializing the source code. In this example, this code starts a web server that will function as the system interface for the automated acceptance tests (1). As soon as the web server is up and running (2), the BDD framework receives a message that the system is initialized (3). This is its trigger to start the actual testing process by parsing the scenario's first test step (4&5). The step *given I go to the "contact" page* is operationalized as `visit path_to(contact_page)` which instructs the BDD automation framework to navigate its browser to `localhost:5000/contact_page` (6). This requests the running web server to execute the source code related to rendering the contact_page view (7&8) and respond with a 200 OK http status code when complete (9). This prompts the second step (10&11) of filling in the `Question`

form field with `'Hello World'` (12). As this is not an action that triggers anything, the BDD framework can immediately progress to the next step (13&14) and clicks the `'Submit'` button (15). This triggers a POST request of the contact form (16) that the source code processes, resulting in the sending of an email to the organizer (18) and the rendering of a view confirming that the form has been submitted (17). The 200 OK http status (19) triggers the BDD framework to proceed with the final step (20&21) of opening the event organizer's inbox (22) and assessing whether it contains an email with `'Hello World'` as its content (24).

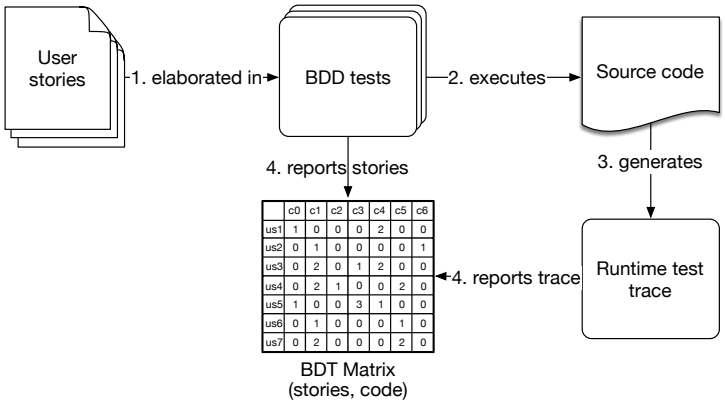
Although BDD is still a fringe development process whose adoption is an order of magnitude smaller than Test Driven Development[1], BDD has evolved and grown considerably in the past ten years. Community-driven initiatives have resulted in a mature development approach with ample reference literature and strong tool support. Nowadays, the Cucumber tool and its associated *The Cucumber Book* are the de-facto standard for BDD (Wynne and Hellesoy, 2012) thanks to their strong adherence to the original BDD philosophy and focus on specifying tests that can easily be read and written by anyone on the team.

Despite BDD's potential, research that uses or extends the approach is still limited. Soeken and colleagues use natural language processing techniques to analyze BDD acceptance tests and suggest source code fragments for implementation such as classes, attributes and operations (Soeken et al., 2012). Similarly, Gao and Li's automatic Problem-to-Design tool generates user scenarios and test code from human-made problem diagrams (Gao and Li, 2016). Another initiative with enduring research interest is reconciling BDD with Business Process Modeling (BPM). De Carvalho et al.'s initial work on mapping BPM constructs to BDD ultimately resulted in Business Language Driven Development, which demonstrates the possibility to use BPM notation as part of BDD's language (de Carvalho et al., 2010). Furthermore, Lübke and Van Lessen describe how they employ BPM Notation 2.0 as a graphical ubiquitous language for BDD which they employ to automatically generate executable BPELUnit tests (Lübke and van Lessen, 2016).

[1] https://goo.gl/Lp0a6Q

7.3 BEHAVIOR-DRIVEN TRACEABILITY

We propose the Behavior-Driven Traceability Method or *BDT* (Fig. 7.2) that automatically establishes ubiquitous traceability on top of the well-established BDD process. BDT relies on two key features of BDD: (i) its detailed decomposition of each user story in brief scenario tests that describe end-user interaction in a stepwise fashion, and (ii) the practice of operationalizing these steps on the system's interface instead of the source code. We explain how the BDT Method takes advantage of these characteristics and introduce the *BDT Tracer* for building the so-called *BDT Matrix* that records the source code called for each user story. We illustrate each step using EventCompany as an example.



	c0	c1	c2	c3	c4	c5	c6
us1	1	0	0	0	2	0	0
us2	0	1	0	0	0	0	1
us3	0	2	0	1	2	0	0
us4	0	2	1	0	0	2	0
us5	1	0	0	3	1	0	0
us6	0	1	0	0	0	1	0
us7	0	2	0	0	0	2	0

Figure 7.2: Behavior-Driven Traceability Method

As an extension of BDD, BDT first requires the formulation of user stories, which the developers subsequently manually *(1) elaborate in* a collection of BDD tests that validate whether the developed features satisfy the customer's requirements. We expand upon EventCompany's initial user story and associated BDD test in Listing 1 by introducing us2: *"As an Organizer, I want to register my event, so that I can sell tickets"*. As the development of both user stories completes, successfully running the BDD test suite *(2) executes* the source code by mimicking how a real person would use the system's interface. To *(3) generate* runtime test traces for the BDD tests we need to integrate with and extend a project's source code so it logs all invoked methods. To achieve this, we leverage the availability of runtime tracers for programming

languages: Python has traceback, .Net comes with Environment.StackTrace and Java supports getStackTrace. Since BDD tests decompose each user story into small steps, these tracers can relate lines of code to BDD artifacts at *three levels of granularity*: an entire user story, a scenario that tests a user story part, or a single scenario step. Our BDT Tracer configures Ruby's TracePoint to trace all relevant method calls in a Ruby on Rails project: goo.gl/SiiRGg.

Listing 2 Snippet of the output of the BDT Runtime Tracer for the Contact Form user story (us1)

	File	Class:method	us#	BDD scenario	BDD test step
1	class/user.rb	User:current_user	us1	feature/contact.feature:5	feature/contact.feature:6
2	class/user.rb	User:track_visitor	us1	feature/contact.feature:5	feature/contact.feature:6
3	class/form.rb	Form:process	us1	feature/contact.feature:5	feature/contact.feature:10
4	class/user.rb	User:track_visitor	us1	feature/contact.feature:5	feature/contact.feature:10

Listing 2 shows a partial runtime trace generated by the Ruby BDT Tracer for the Contact Form feature of Listing 1. Each line lists the location of the executed source code file, the invoked class and method, the user story in the description, the BDD scenario and the corresponding BDD test step that triggered the executed source code. Note that the numbers 5, 6 and 10 after the BDD scenarios and test steps refer to the line numbers of Listing 1. This is formalized in Definition 1.

Definition 1 [Runtime test trace] Given a set of user stories $US = \{us_1, \ldots, us_n\}$, a runtime test trace rtt_{us} is a list built by sequentially executing all the BDT tests of all user stories in the set, where every list element is a tuple $\langle loc, cl, meth, us, scen, step \rangle$ such that *loc* is the source code location of the class *cl* whose method *meth* was executed as part of *step* of the scenario *scen* of the user story *us*.

The BDT method then *(4) combines the reported* runtime test trace with the BDD test's annotated user stories into the BDT Matrix. This matrix records user stories on the Y axis, source code methods on the X axis and the number of times each user story invokes each method in the cells. Table 7.1 shows the BDT Matrix for the two EventCompany's user stories introduced earlier in this paper. Definition 3 formalizes the notion of a BDT Matrix after Definition 2 presents the preliminary concept of invocation frequency.

Definition 2 [Invocation frequency] Given a user story us_i and a runtime trace $rtt_{\{us_i\}}$, the invocation frequency for a method m of class c (denoted as $c{:}m$) $invFreq_{us,c:m} \in \mathcal{N}$ indicates the number of tuples $\langle loc, cl, meth, us, scen, step \rangle$ in $rtt_{\{us_i\}}$ such that $cl = c$ and $meth = m$.

Definition 3 [BDT Matrix] Let $US = \{us_1, \ldots, us_n\}$ be a set of user stories, $M = \{m'_1, \ldots, m'_q\}$ be a set of methods (each included in some class). A BDT Matrix BDT has size $n \times q$, and each cell indicates the invocation frequency of a method in the BDD tests of a user story. Formally, $\forall i, j \in \mathcal{N}^+$ such that $i \leq n, j \leq q$, then $BDT_{i,j} = invFreq(us_i, m'_j)$.

When a software development team creates individual BDD tests for each user story, applying BDT results in a BDT Matrix that allows a developer to request all the source code invoked to realize a given user story. By applying smart filtering techniques the BDT Matrix can then be used to produce a variety of reports, such as methods that are never called in the entire test suite to identify dead code, or all the classes involved in a specific user story to inform developers modifying or refactoring a user story's code.

Table 7.1: Partial BDT Matrix for the EventCompany Case

	User:		**Form:**		**Event:**
	current_user	track_visitor	process	encode_media	check_pricing
us1	1	2	1	0	1
us2	1	0	1	1	1

Table 7.2: Normalized Partial BDT Matrix for the EventCompany Case

	User:		**Form:**		**Event:**
	current_user	track_visitor	process	encode_media	check_pricing
us1	0.5	2	0.5	0	0.5
us2	0.5	0	0.5	1	0.5

Note, however, the significant overlap in the BDT Matrix: 3 out of 5 methods are called when running the test for both user stories. This is typical for high quality software that relies on code reuse. Thus, BDT is prone to including omni-present code: classes and methods that are called in almost every test step. These create noise in the output, making it more difficult to identify the unique code for that user story.

To reduce this effect, we take inspiration from *feature location* techniques that automatically identify which part of the source code implements a given functionality (Dit et al., 2013). Our situation is comparable to *software reconnaissance* (Wilde and Scully, 1995): runtime traces produced by running test scenarios contain a lot of shared source code, thereby hiding uncommon and feature-specific methods. They identify the *"uniquely involved components"* for a feature by taking the set of components exercised as part of the test cases related to that feature and then excluding any components exercised in test cases unrelated to the feature. We go beyond this approach and look at the relative importance of the uniquely involved components.

To identify all uncommon, relevant methods and retain their relative importance, we normalize the BDT Matrix by dividing the number of method calls for a user story by the total number of user stories, scenario's or steps the method is called in, resulting in the normalized BDT Matrix in Table 7.2. Note for example that the User:current_user method is called in both user stories, resulting in a normalized BDT output of $1/2 = 0.5$. User:track_visitor is only called in us1, however, leading to a normalized BDT output of $2/1 = 2$, emphasizing the important role of this method for us1.

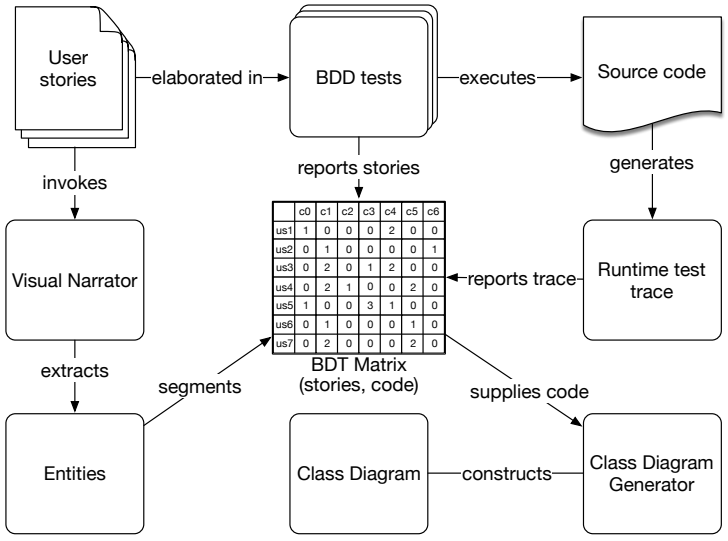
Definition 4 [Normalized BDT Matrix] Given a BDT Matrix of size $n \times q$, a normalized BDT Matrix $NBDT$ has size $n \times q$ and its cells denote the relative frequency for a user story to invoke a method with respect to how many other user stories also invoke that method. Formally, let ite be the if-then-else operator, and $\forall i, j \in \mathcal{N}.i \leq n, j \leq q$,

$$NBDT^{us}_{i,j} = \begin{cases} 0 & \text{if } BDT_{i,j} = 0 \\ \dfrac{BDT_{i,j}}{\sum\limits_{1 \leq k \leq n} ite(BDT_{k,j} > 0, 1, 0)} & \text{otherwise} \end{cases}$$

It is also possible to obtain a more fine-grained normalization with respect to the number of test scenario ($NBDT^{sc}$) or steps ($NBDT^{st}$), instead of the number of user stories. In those two variants, the numerator stays the same ($BDT_{i,j}$), while the denominator is the number of scenarios and the number of steps that invoke method j, respectively.

7.4 BDT CHANGE IMPACT ANALYSIS

The BDT Change Impact Analysis Method (BDT CIA or CIA) uses an entity extraction tool to dynamically construct BDT matrices based on only those user stories that contain shared entities with a new user story. The resulting matrix serves as input to generate reports that stakeholders use to establish the change impact of the new user story: the *Change Impact Matrix* (CI Matrix). Furthermore, they can be used to generate specific class diagrams that includes only the identified classes and methods. Note that raw BDT output of an implemented user story already suffices to determine the change impact of deleting it, for it includes all methods which may have to be changed or removed. The Change Impact Matrix is formally presented in Definition 5.



	c0	c1	c2	c3	c4	c5	c6
us1	1	0	0	0	2	0	0
us2	0	1	0	0	0	0	1
us3	0	2	0	1	2	0	0
us4	0	2	1	0	0	2	0
us5	1	0	0	3	1	0	0
us6	0	1	0	0	0	1	0
us7	0	2	0	0	0	2	0

Figure 7.3: Overview of the BDT Change Impact Analysis Method

Definition 5 [Change Impact Matrix] Given a BDT Matrix BDT of size $n \times q$ and a user story us a Change Impact Matrix CIM_{us} is a matrix of size $p \times q$ with $p \leq n$ that includes invocation frequency only for those user stories whose text shares some concepts with the concepts of us. Let $conc_{us}$ indicate the set of concepts in the text of us. Then, CIM_{us} is a row slicing of BDT such that:

$$\forall j \in [1, n].BDT[j] \in CIM_{us} \leftrightarrow conc(us_j) \cap conc(us) \neq \emptyset$$

The BDT CIA Method in Fig. 7.3 extends the BDT Method with the Visual Narrator tool (Robeer et al., 2016) for user story entity and relationship extraction. Let us add a new user story US3 to EventCompany's user stories: *"As a Customer, I want to purchase tickets, so that I get access to an event"*. To assess the change impact of US3, we first construct the BDT Matrix in Table 7.1 by executing the BDD tests for US2 and US1.

Next, BDT invokes Visual Narrator to *extract* all entities from the existing user stories as well as the new user story. The BDT matrix is then row sliced into the CI Matrix by including only the rows whose user story shares concepts with the new user story. For example: US3 matches with US2 as they both include the entities ticket and event. Consequently, BDT CIA includes only the classes and methods in US2 as those that the new US3 is likely to impact. To highlight the impact of the new story, a normalized version of the CI Matrix can be generated by applying the normalization scheme presented in Definition 4. Finally, the class diagram generator constructs a hyper-specific view that only includes US2's classes and methods (Fig. 7.4). Note this does not include a `Ticket` class because US2 did not yet necessitate its implementation.

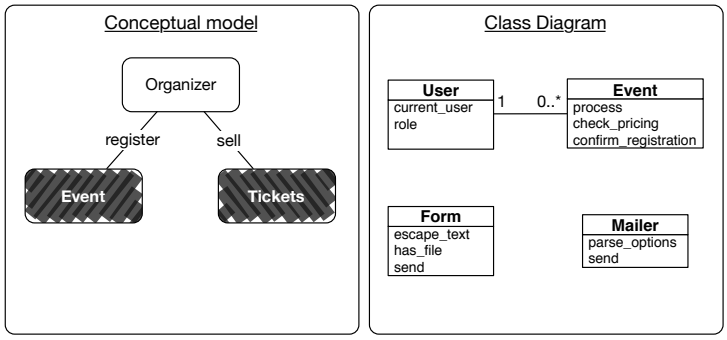


Figure 7.4: Dynamically generated Class Diagram of the US2 story

7.5 EVALUATING BDT AND BDT CIA

To evaluate the feasibility of our approach we developed a functional prototype that automates both the BDT Method and the BDT CIA Method and made it available on GitHub (https://github.com/gglucass/BDT). We apply the prototype to the open source project *Archive of Our Own* to answer the three evaluation questions Q1–Q3.

Archive of Our Own (AO3, https://archiveofourown.org) is an initiative by the Organization for Transformative Works that enables authors of so-called *fanfiction* to store and share their work. The community has over one million users who have shared nearly 3 million stories. We apply the prototype to AO3's project status on October 23rd 2015 (goo.gl/FNRwnn). On this day, the project's automated acceptance tests consist of 115 separate feature files, 75 of which are annotated with a user story. Altogether, the project contains 710 test scenarios with 10,662 test steps.

7.5.1 *Accuracy and relevance of the traces (Q1)*

Executing AO3's entire BDD test suite with the BDT prototype results in BDT Tracer output of 8,757,057 lines. Of these, just 1,941,982 or 22.18% are calls to new code created as part of the AO3 project, the other 77.82% are all calls to methods of 3rd party plugins or the Ruby on Rails framework. The project-specific lines call 917 unique methods in 99 distinct class files: much less than the 1,328 unique methods within the 145 class files of the AO3 project: a method recall of 69.05% and class recall of 68.28%.

The BDT tracer misses 26 out of 46 uncalled classes as these do not define any methods for the system to call; their only purpose is to configure a plugin and/or enable CRUD database actions. Examples are (i) the `Admin` class which configures the *Authlogic* plugin and (ii) the `LogItem` class which merely facilitates CRUD calls to the Ruby on Rails database plugin ActiveRecord. That leaves 20 classes that *do* define methods, yet are never called as part of the tests: a class recall of 86.21%. Note that of these 20, one contains a superfluous method that is not called anywhere in the code, another is a super class whose subclasses *are* called and 18 classes with 153 methods are only called as part of one-time setup scripts or periodical jobs. As these do not concern *user behavior*, the class recall can be considered 100% and method recall becomes 78.04%.

Using BDT Tracer output, we build a BDT matrix; since showing the entire matrix is not possible in paper form, we examine four reports built from it:

– Summed class calls for all BDD tests (Table 7.3);
– Top 20 invoked methods for all tests (Table 7.4);
– Class calls for two individual user stories (Table 7.5);
– Top 10 method calls for those two stories (Table 7.6).

The first two reports highlight the dominant role of the top six classes in Table 7.3 which together are responsible for 70.51% of the method calls and contribute 19 out of 20 top methods in Table 7.4. These classes and methods are such a central part of the software system that they are involved in almost every user interaction. On the other end of the table, the bottom 74 classes account for just 2.73% of all calls. This disparity demonstrates the necessity for normalizing the raw BDT output to obtain actionable reports (and justifies the notion of NBDT), which we explore in the next subsection.

The reports in Table 7.5 and 7.6 are built from the BDT matrix for the BDD tests associated with the user stories (US1) *"As a user, I want to add my items to collections"* and (US2) *"As an author, I want to be able to delete a comment a reader added to my work In order to remove a comment from public view"*. Examining their data reveals three characteristics of BDT output that, again, call for normalization:

1. *Universal Classes.* There is substantial overlap in the called classes and methods. Out of the 32 classes in Table 7.5, 24 classes are present in both lists. Similarly, the 6 non-underlined methods in Table 7.6 occur in both method top 10s. In fact, those classes and methods are called at least once in (nearly) all features.
2. *Relevancy.* Despite their overlap, each report includes unique classes and methods relevant to their respective user stories. US1 contains 4 unique classes `Collection`, `CollectionItem`, `CollectionParticipant` and `Bookmark`, while US2 has 2 `Comment` and `CommentObserver`. Moreover, the 4 underlined methods barely or never occur in other stories' outputs.
3. *Class Distribution.* The number of calls for each class is similar. The by far most called classes of both lists are `Work`, `Tag` and `User`, together with the unique class for that user story: `Collection` or `Comment`. This distribution similarity is due to the central role of those classes in the AO3 website: a place where *users* exhibit their *work* based on existing pop culture which they *tag* to share with other interetsed users.

Despite the evidenced need for normalization, the high class recall of 86.21–100% for AO3 and the identification of user-story-specific classes and methods suggest a positive evaluation of *"Q1. How accurate and relevant are the generated traces?"*

Table 7.3: BDT report 1: summed class calls for all AO3 tests

Class	No	Class	No
Tag	399565	ChallengeClaim	5324
User	266013	CommonTagging	5114
Work	248239	RedisSearchIndexQueue	4503
Collection	243128	Preference	3990
Prompt	112018	CollectionItem	3210
WorkSearch	100273	InboxComment	2612
TagSet	69870	ChallengeAssignment	2476
Pseud	67804	CacheMaster	1716
Skin	56454	Offer	1430
OpenStruct	51246	Reading	1422
PromptRestriction	43368	SkinParent	1420
Chapter	37452	PotentialMatch	1416
Comment	32992	BookmarkSearch	1379
Media	27221	CommentObserver	1227
ChallengeSignup	24681	Gift	1139
CreationObserver	18277	TagNomination	1134
Bookmark	14859	SearchResult	1090
Series	13196	TagSetNomination	1072
OwnedTagSet	11825	StoryParser	880
Tagging	11699	Subscription	816
AdminSetting	10372	CharacterNomination	767
Fandom	8987	ResponseObserver	747
Request	7635	PotentialPromptMatch	728
GiftExchange	5974	MetaTagging	672
PromptMeme	5890	Rating	606

Class	No	Class	No
CollectionParticipant	528	AdminBanner	44
Language	517	CollectionPreference	44
StatCounter	418	CollectionPreferenceObserver	44
CastNomination	401	AbuseReport	42
People	304	ChallengeSignupSummary	40
CacheMaster	286	Search	37
IndexQueue	272	KudoObserver	30
Character	268	ChallengeSignupTagSummary	28
FandomNomination	256	FannishNextOfKin	27
Invitation	244	FavoriteTag	27
SerialWork	239	ExternalAuthorName	24
Relationship	232	ExternalCreatorship	24
AdminPost	195	Warning	22
Freeform	182	UserInviteRequest	17
TagSetAssociation	180	Feedback	16
PotentialMatchSettings	162	AdminActivity	15
Kudo	154	Profile	10
ArchiveFaq	145	InviteRequest	8
ExternalWork	126	WorkObserver	8
FilterTagging	105	TagSearch	5
RedisMailQueue	87	AdminPostTag	2
RelatedWork	73	PseudSearch	2
WorkSkin	62	Locale	1
ExternalAuthor	52	WranglingGuideline	1
Creatorship	48		

Table 7.4: BDT report 2: summed method calls for all AO3 tests

Class	Method	No
Tag	autocomplete_prefixes	96786
User	to_param	85821
Tag	find_by_name	65874
Collection	to_param	48649
Tag	find_or_create_by_name	38631
Prompt	respond_to?	35155
Work	title	30323
Media	uncategorized	27221
User	is_author_of?	25843
User	active?	20955
User	is_tag_wrangler?	20190
Comment	ultimate_parent	18921
Tag	set_sortable_name	16673
Tag	check_synonym	16673
Tag	squish_name	16673
Tag	unwrangleable_status	16673
WorkSearch	set_tag_fields!	16161
WorkSearch	set_parent_fields!	16161
WorkSearch	clean_up_angle_brackets	16161
WorkSearch	set_language!	16161

Table 7.5: BDT report 3: class calls for AO3's US1 & US2

US1. Collection Items		US2. Delete Comments	
Class	**No**	**Class**	**No**
Work	6398	Work	2926
Tag	5480	Tag	2365
Collection	3990	User	2185
User	3122	Comment	588
WorkSearch	1146	Pseud	496
Pseud	820	Chapter	476
Chapter	717	WorkSearch	468
Skin	636	OpenStruct	282
OpenStruct	581	Skin	274
CreationObserver	530	Media	140
Bookmark	426	CreationObserver	140
Media	330	Tagging	120
Tagging	279	CommentObserver	50
CollectionItem	229	Fandom	48
Fandom	108	AdminSetting	34
AdminSetting	94	CommonTagging	32
CommonTagging	72	Reading	26
CacheMaster	56	ResponseObserver	24
Preference	54	Preference	24
RedisSearchIndexQueue	30	InboxComment	16
Reading	22	Language	4
CollectionParticipant	21	Rating	4
InboxComment	20	StatCounter	4
Rating	13	Subscription	4
Subscription	10	RedisSearchIndexQueue	4
Language	9		
ResponseObserver	9		
StatCounter	9		

Table 7.6: BDT report 4: method calls for AO3's US1 & US2

US1. Collection Items			US2. Delete Comments		
Class	**Method**	**No**	**Class**	**Method**	**No**
Tag	autocomplete_prefixes	1548	Tag	autocomplete_prefixes	688
Work	title	904	User	to_param	461
User	to_param	874	User	is_author_of?	387
Tag	find_by_name	760	Work	title	351
Collection	to_param	504	Tag	find_by_name	312
Collection	closed?	487	Comment	ultimate_parent	274
Tag	find_or_create_by_na	445	Work	anonymous?	208
Collection	autocomplete_prefixes	392	Tag	find_or_create_by_na	180
User	is_author_of?	349	User	default_pseud	160
Media	uncategorized	330	User	is_tag_wrangler?	144

7.5.2 *Normalization to improve relevance (Q2)*

Due to the many omni-present methods, US1's and US2's BDT output is highly similar. To highlight their differences we employ the normalization method of Section 7.3, Definition 4: we execute a row slicing of the BDT Matrix based on the number of scenarios and we derive $NBDT^{sc}$.

Given AO3's 710 scenarios, the average calls for the universal classes `Tag`, `User` and `Work` in Tables 7.3 are 563, 375 and 350, respectively, while for the methods `Tag:autocomplete_features`, `User:to_param` and `Work:title` they are 136, 121 and 43. Applying these ratios produces the reports of Table 7.7 and 7.8; they are less similar and the specific classes have taken over the prominence of the universal classes. Four of five top classes for both user stories are highly relevant: `CollectionItem`, `CollectionParticipant`, `CreationObserver` and `Bookmark` for US1 and `CommentObserver`, `Comment`, `ResponseObserver` and `Chapter` for US2. Even `Work`'s sixth position is appropriate, for users comment on works and group them in collections. Moreover, the universal classes `Tag` and `User` are no longer at the top and the common yet irrelevant classes `Skin` and `WorkSearch` are at the bottom of the report.

Yet, the normalization is not perfect: `Collection` is underrepresented, while `CacheMaster` and `Reading` are ranked too high. While `Collection` is important for US1 and appears in 15.8% of method calls, it stands in the middle of the normalized report. On the other hand, `CacheMaster` takes the third position in the normalized class report for US1, despite its irrelevance to US-1's core functionality: after creating, updating or deleting a collection item or work it resets the search indexes. Similarly, the third class in the Delete Comments report `Reading` is not unique nor relevant to US2: it is used to log the user's reading history. Both issues are due to the oversensitivity of our

Table 7.7: BDT Report 5: normalized class calls for AO3's US1 & US2

US1. Collection Items		US2. Delete Comments	
Class	**%**	**Class**	**%**
CollectionItem	13.21	CommentObserver	16.32
CollectionParticipant	7.36	ResponseObserver	12.87
CacheMaster	7.27	Reading	7.32
CreationObserver	5.37	Comment	7.14
Bookmark	5.31	Chapter	5.09
Work	4.77	Work	4.72
Tagging	4.42	Tagging	4.11
StatCounter	3.99	StatCounter	3.83
Rating	3.97	User	3.29
Chapter	3.54	Language	3.10
Language	3.22	CreationObserver	3.07
Collection	3.04	Pseud	2.93
Reading	2.86	Rating	2.64
CommonTagging	2.61	CommonTagging	2.51
Tag	2.54	InboxComment	2.45
Preference	2.51	Preference	2.41
Subscription	2.27	Tag	2.37
Media	2.24	OpenStruct	2.20
Pseud	2.24	Fandom	2.14
ResponseObserver	2.23	Media	2.06
Fandom	2.23	Subscription	1.96
User	2.17	Skin	1.94
WorkSearch	2.12	WorkSearch	1.87
OpenStruct	2.10	AdminSetting	1.31
Skin	2.09	RedisSearchIndexQueue	0.36
AdminSetting	1.68		
InboxComment	1.42		
RedisSearchIndexQueue	1.23		

Table 7.8: BDT Report 6: normalized method calls for AO3's US1 & US2

US1. Collection Items			US2. Delete Comments		
Class	**Method**	**%**	**Class**	**Method**	**%**
Collection	autocomplete_prefixes	1.45	Comment	comment_owner	2.54
Collection	closed?	1.13	Comment	top_level?	2.47
Work	visibility_changed?	0.90	Work	last_posted_chapter	2.15
Work	bust_anon_caching	0.90	CommObs.	notify_user_..?	1.92
Work	rm_outdated_download	0.90	Work	chapters_in_order	1.72
Work	download_dir	0.89	Reading	update_or_create	1.39
Work	adjust_series_restriction	0.88	Work	guest_kudos_count	1.35
Work	check_for_inv_chapters	0.80	Tag	string	1.29
Work	check_filter_counts	0.80	Work	visible?	1.24
Work	sorted_title	0.77	Work	chapter_total_disp	1.12

normalization. `Collection` cannot exceed the high 342 division (the $\sum$ in the denominator of Definition 4), obfuscating its role. `CacheMaster` and `Reading` are advantaged for they appear an above average number of times in a small number of features. Their resulting low division of around 2 inevitably brings them to the top for those features' US output.

However, the normalized reports' are generally good at highlighting the most highly relevant classes and methods, thereby suggesting a positive evaluation of *"Q2. Does normalizing the traces influence its relevance?"* As discussed in Section 7.6, further work is however necessary to improve the normalization and avoid oversentitivity.

7.5.3 *Effectiveness of Change Impact Analysis (Q3)*

The evaluation presented so far is based on AO3's situation on October 23rd 2015. A day later, October 24th 2015, introduced a new user story US3: *"As an author, I'd like to be able to moderate comments in order to avoid spam and troll comments"*. The update includes new source code to satisfy the user story and a BDD test consisting of 7 scenarios with 81 steps (goo.gl/dk1Uew). We use this scenario to evaluate the accuracy of the BDT CIA method.

To automatically identify the change impact of US3, we can reuse the BDT Matrix created in the previous subsections, and we apply Visual Narrator to extract and match the entities of the new user story to the entities in the existing user story collection. The entities in US3 are: *author*, *comment*, *spam* and *troll*. Out of the 75 user stories in the AO3 project, 17 include the entity *author*, 3 include *comment* and 1 includes *both* author and comment: our previous example user story on Delete Comments. This means we can reuse our previous BDT reports on US2 for our BDT CIA evaluation.

Applying the BDT prototype to US3's BDD Tests produces the reports shown in Table 7.9 and Table 7.10. In total, these tests make 12,941 calls between 27 unique classes. Many of these calls go to similar classes and methods as per US2's BDT reports in Table 7.7 and 7.8: `Tag`, `Work` and `User` dominate the calls, while the central class `Comment` is in the semi-top for both. While their ordering is different, the top 8 methods occur in both reports, with only the bottom two `User` methods in US2 replaced by `Media` and `Tag` in US3. Applying Pearson's correlation coefficient to the BDT reports results in $\rho_c = 0.9777$ (for classes) and $\rho_m = 0.9612$ (for methods) with $p_{cm}<0.001$: a strong positive correlation.

Table 7.9: BDT Report 7: class calls for AO3 US3 moderate comments

Regular		Normalized	
Class	**No**	**Class**	**%**
Tag	3653	CommentObserver	12.09
Work	3165	ResponseObserver	9.06
User	2232	Tagging	5.79
WorkSearch	666	Reading	5.44
Pseud	583	StatCounter	5.40
Comment	461	Comment	4.50
Skin	451	Language	4.36
Chapter	432	Work	4.11
OpenStruct	373	Rating	3.72
Tagging	210	Chapter	3.72
Media	199	Kudo	3.63
CreationObserver	126	CommonTagging	3.53
Fandom	84	Fandom	3.01
AdminSetting	66	Tag	2.95
CommonTagging	56	Pseud	2.77
CommentObserver	46	Subscription	2.77
Preference	33	InboxComment	2.72
Reading	24	User	2.70
InboxComment	22	Preference	2.67
ResponseObserver	21	Skin	2.58
Kudo	8	Media	2.36
Rating	7	OpenStruct	2.35
Language	7	CreationObserver	2.22
StatCounter	7	WorkSearch	2.14
Subscription	7	AdminSetting	2.05
KudoObserver	2	KudoObserver	0.91
RedisMailQueue	1	RedisMailQueue	0.45

Table 7.10: BDT Report 8: method calls for AO3 US3 moderate comments

Regular			Normalized		
Class	**Method**	**No**	**Class**	**Method**	**%**
Tag	autocomplete_prefixes	1204	Work	last_posted_chapter	1.92
User	to_param	490	CommObs	notify_user_..?	1.89
Tag	find_by_name	419	Comment	top_level?	1.82
User	is_author_of?	345	Comment	comment_owner	1.79
Work	title	345	Work	guest_kudos_count	1.46
Work	anonymous?	254	Work	chapters_in_order	1.44
Tag	find_or_create_by_na	234	Tag	string	1.39
Comment	ultimate_parent	219	Work	find_all_comments	1.35
Media	uncategorized	199	Work	visible?	1.34
Tag	set_sortable_name	196	Chapter	moderated_com..?	1.22

The normalized report for US3 shows a similar correspondence to US2. At the top of both reports are `CommentObserver` and `RespondentObserver` and the sub-top includes both `Reading` and `Comment`. Again, the top 8 methods occur in both reports. The normalized method calls of US3 contains two new methods (`Chapter:moderated_commenting_enabled?` and `Work:find_all_comments`) that replace two methods of US2

(`Reading:update_or_create` and `Work:chapter_total_display`). Applying Pearson's correlation coefficient to the normalized BDT report results in $\rho\_c = 0.9286$ and $\rho\_c = 0.8188$ with $p\_cm<0.001$: again, both are strong positive correlations.

By design, the BDT CIA method is incapable of fully accurately predicting the change impact for a new user story. Nevertheless, the similarity of the BDT reports—confirmed by Pearson's correlation coefficient—shows the ability of identifying relevant source code that is likely to be involved in satisfying a new user story. As such, based on this first evaluation for the AO3 case, we provide a partial positive answer to *Q3*.

Our study of BDT CIA's ability to identify which classes and methods should be modified to satisfy the new user story is less positive. Looking at the contributed source code for US3, the additional class code consists of 13 new lines in `Comment`, 3 in `Chapter`, 26 in `CommentsController`, 5 in `ChaptersController`, 25 new lines between 4 `Comment` views, 6 new lines in 2 `Work` views and 2 new lines in the `Help` view. Altogether, these new source code lines make the following number of method calls: 21 calls to methods of `Comment`, 17 for `Chapter`, 3 for `User` and 2 for `Work`. The classes with the most code additions and new calls to methods are `Comment` and `Chapter`. While these classes are in the fourth and fifth position of the normalized BDT output for US2, they are not nearly as prominent as `CommentObserver` and `ResponseObserver`. These two are definitely also part of the call trace to satisfy the new user story, but are not directly invoked in the new source code. The top 10 called methods include just one `Comment` method: `ultimate_parent`. This is, however, the most called method in the new source code. `Work` and `User` are sixth and ninth on the normalized BDT report for US2.

This analysis shows there is little actual overlap between the BDT CIA output and the new code that satisfies it. The little overlap there is, is overshadowed by the classes and methods that do not need to be called nor taken into account when developing the new user story. This indicates that BDT CIA output is not actionable for identifying the exact classes or methods that need to be extended. Altogether, our two analysis of BDT CIA output prompt us to negatively answer *"Q3: Do change impact analysis reports adequately reflect the consequences of new requirements?"*

7.6 DISCUSSION AND OUTLOOK

With the aim of establishing *ubiquitous* traceability (Cleland-Huang et al., 2014a), we proposed two methods built on the agile software development process *Behavior Driven Development*: the Behavior Driven Traceability Method (BDT) and the BDT Change Impact Analysis Method (BDT CIA). We have shown how to trace user story requirements to source code by processing runtime traces of automated acceptance tests, and how normalized versions of the BDT matrix emphasize user-story-specific classes and methods.

Initial testing of our prototype on the open source project *Archive of Our Own* (https://archiveofourown.org) produces promising accuracy and relevance results that motivates us to continue investigating this domain. Both raw and normalized BDT output have merit: the former captures the prevalence and importance of a small number of classes and methods, while the latter highlights classes and methods that are important to a single user story. We obtained less positive results about BDT CIA's ability to identify classes and methods that will be changed to satisfy a new user story, at least in the context of our case study.

Threats to Validity. *External validity* threats reduce results generalizability. Our methods only apply to teams that create software using BDD, not yet a mainstream technique and primarily used in web development. Furthermore, the AO3 project used in the evaluation was chosen for its suitability from a list of open source projects. Often, open source projects do not cover all functionality making BDT output less precise and useful. *Construct validity* threats concern the degree to which a test measures what it intends to measure. While the evaluation closely examined BDT output and evaluated its conformance to source code, we did not evaluate whether practitioners find the output useful in establishing traceability or analyzing change impact. *Internal validity* threats focus on the experiments conduction. Comparative analyses of the BDT output and analysis of source code to satisfy US3 were conducted manually and are thus subject to human error.

Future work. In future work we intend to evaluate BDT output's accuracy by testing our prototype on industry projects and by involving practitioners. In particular, we want to understand whether the classes and methods in the NBDT matrix are relevant for software engineering tasks such as refactoring and resolving bugs. We are currently applying BDT and BDT CIA at one industry partner to demonstrate to the Central Dutch Bank that its ability to accurately pinpoint which code is relevant for which requirement at any given time.

Conclusion

8

Motivated by the lack of academic work on user stories and the unique opportunities offered by user stories' simple and uniform structure, this dissertation posed one main research question:

MRQ — *How to employ computational linguistic techniques to understand user story requirements?*

To formulate an answer, Chapters 2-7 each investigated specific problems and opportunities related to user stories. Together, these chapters established how to employ computational linguistic techniques for three purposes that increase user story understanding among practitioners:

(i) detecting defects in user stories and suggesting corrective actions

(ii) extracting entities and relationships from user stories to construct conceptual models

(iii) reducing the complexity of a conceptual model visualization with zooming and filtering

To enable practitioners to take advantage of these findings, Chapters 3, 5 and 6 presented three tools that correspond to these purposes: AQUSA, Visual Narrator and Interactive Narrator. In Chapter 5 we proposed the Grimm User Story Method which combines these tools into a holistic approach for requirements engineering based on user stories. The Grimm method is the primary contribution of this dissertation and our response to the low quality of user stories in industry. Fig 8.1 illustrates how the Grimm method combines our tools in such a way to improve user story quality and stimulate discussion on user stories.

After a stakeholder creates an initial set of user stories, each tool supports a human requirements engineer in reaching these goals in some way: (1) AQUSA validates the user stories' quality by automatically detecting defects using NLP techniques, (2) Visual Narrator extracts entities and relationships from the user story collection to construct a conceptual model that facilitates detecting incompleteness and inconsistencies, and (3) Interactive Narrator combines the conceptual model with other data sources to generate specialized visualizations

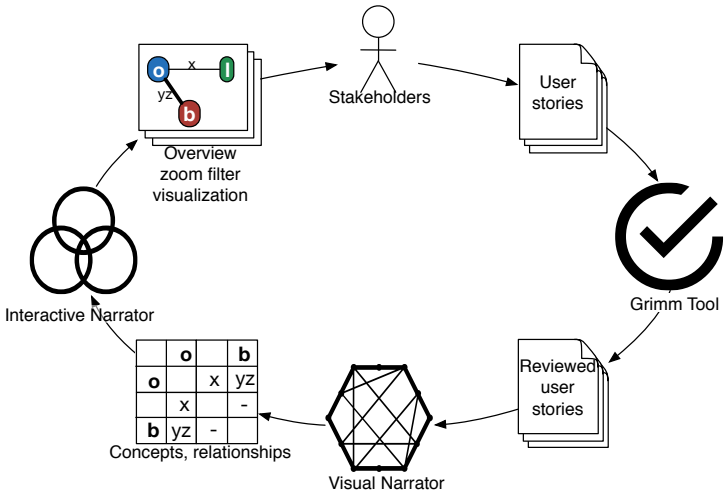


Figure 8.1: The Grimm User Story Method for agile requirements engineering

that zoom in or filter on user story element(s) to facilitate focused discussion on ambiguities, dependencies and inconsistencies. Stakeholders then use these different visualization views as input for collaboratively resolving issues by modifying existing user stories or identifying new ones. This triggers a new Grimm method iteration in Figure 8.1, which repeats until all issues are resolved.

Although our evaluation of the Grimm method's impact on work deliverable quality and productivity in Chapter 4 did not produce statistically significant conclusions, the reported increase in intrinsic user story quality and conversation surrounding user stories indicate that applying computational linguistic techniques indeed supports practitioners in understanding user stories. Aside from improving existing user story processes, the Grimm method also prepares practitioners to reap additional benefits. The Behavior Driven Traceability Change Impact Analysis Method presented in Chapter 7 takes advantage of Visual Narrator output to determine new user stories' impact on existing source code.

To reach this answer and contribute the Grimm method, this dissertation posed six distinct research questions. In the next pages we briefly discuss the answers to each research question and draw six conclusions. Together, these form and inform our answer to the main research question.

RQ1 — *How do industry practitioners use and perceive user stories?*
Despite the widespread adoption of user stories in industry, little is known about how practitioners actually work with user stories. Based on an analysis of 182 survey responses and 21 follow-up interviews, we found that the majority of practitioners perceive user stories to increase work deliverable quality and productivity. After qualitatively analyzing their answers, we reported 12 key findings in Section 2.7. From these, we distill the following three highlights to inform our first conclusion below:

1. Practitioners employ the well-known Connextra user story template, *"As a, I want to, [so that]"*, in combination with the Scrum software development method.

2. User stories enable developing the *right* software, especially when their authors include the *why* part.

3. Adopting INVEST quality guidelines significantly increases the perceived beneficial impact of user stories on productivity and the impact of templates on work deliverable quality.

Conclusion I
Practitioners agree, user stories improve work deliverable quality and productivity. Even the simplest approach to user stories already produces perceived benefits. When practitioners invest time and resources to gain a deeper understanding of user stories, such as by studying the INVEST framework, practitioners perceive an even greater positive impact of working with them.

RQ2 — *What are the textual characteristics of high-quality user stories?*
The quality of user stories created by practitioners varies widely. Through critical analysis of hundreds of user stories and a review of traditional requirements quality frameworks' applicability to user stories, we proposed the **Q**uality **U**ser **S**tory Framework in Chapter 3. It defines 13 criteria for high-quality user stories that practitioners should strive to adhere to:

1. **Well-formed** - A user story includes at least a role and a means
2. **Atomic** - A user story expresses a requirement for exactly one feature.
3. **Minimal** - A user story contains nothing more than role, means and ends.
4. **Conceptually sound** - The means expresses a feature and the ends expresses a rationale.
5. **Problem-oriented** - A user story only specifies the problem, not the solution to it.
6. **Unambiguous** - A user story avoids terms or abstractions that lead to multiple interpretations.
7. **Conflict-free** - A user story should not be inconsistent with any other user story.
8. **Full sentence** - A user story is a well-formed full sentence.
9. **Estimatable** - A story does not denote a coarse-grained requirement that is difficult to plan and prioritize.
10. **Unique** - Every user story is unique, duplicates are avoided.
11. **Uniform** - All user stories in a specification employ the same template.
12. **Independent** - The user story is self-contained and has no inherent dependencies on other stories.
13. **Complete** - Implementing a set of user stories creates a feature-complete application, no steps are missing.

Trying to apply this framework to over 1,000 user stories from 18 organizations, we found that it is difficult to assess whether a user story adheres to many of these criteria without being aware of the organization's context. For the five criteria that could be applied without this knowledge, we identified many violations. These could have easily been prevented if their creators had a better understanding of high-quality user stories.

Conclusion II
It is difficult to verify the Quality User Story Framework characteristics' validity for real-world user stories. When unfamiliar with a project's context, it is particularly difficult to detect those characteristics that necessitate semantic understanding of the terms used in the user story.

Although textual characteristics are a good indicator for the quality of a user story, it is challenging to detect issues that necessitate semantic understanding of the terms used in the user story. This makes it difficult to verify all of the Quality User Story Framework characteristics' validity in real-world user stories.

RQ3 — *How can natural language processing support formulating high-quality user stories?*

To support practitioners in creating high quality user stories, Chapter 3 also introduced the **A**utomatic **Q**uality **U**ser **S**tory **A**rtisan tool. AQUSA employs state-of-the-art NLP techniques to automatically detect violations of a selection of the quality criteria in the QUS framework. Applying AQUSA to over 1,000 user stories from 18 organizations demonstrates the feasibility of employing natural language processing to support formulating high quality user stories: AQUSA is capable of detecting a subset of the QUS framework's violations with 92.1% recall and & 77.4% precision. Note that AQUSA fulfills the desired Perfect Recall Condition for 5 cases, obtains between 90-100% defect recall for 6 sets and manages to get between 55% and 89% recall for the remaining 6. Despite these positive results, in order to determine whether AQUSA's nearly perfect recall is sufficient to produce meaningful benefits, we needed to investigate the impact of introducing AQUSA in real-world software development.

To this end, Chapter 4 studied the effect of applying the Grimm Method's QUS framework and the AQUSA tool into existing user story practices through a multiple case study at three companies. Although the number of user story quality defects decreased by 43.14% after applying the treatment, participants did not perceive a meaningful change in user story quality. Furthermore, while respondents did not believe that QUS and AQUSA contributed to increasing work deliverable quality and productivity, the respondents agreed that communication frequency and effectiveness did improve. These discrepancies highlight the difficulty of measuring the impact of the relatively small change of introducing quality guidelines for requirements in the complex process of software development. The benefits of improving user story quality may unearth themselves much later, or may not even be measurable with the standard agile software development metrics we employed.

Conclusion III

While the Automatic Quality User Story Artisan is capable of detecting violations of some QUS framework characteristics with reasonably high accuracy, the benefit of leveraging natural language processing to support formulating high-quality user stories remains unclear. Incorporating AQUSA in a software development team reduces the number of user story quality violations, yet practitioners do not perceive this positive impact.

RQ4 — *How to identify the principal concepts of a user story collection?* Although easy to read, a textual user story collection does not readily highlight key entities and their relationships such as dependencies and conflicts. In Chapter 5 we presented the Visual Narrator tool that relies on the off-the-shelf NLP toolkit spaCy to orchestrate a selection of state-of-the-art heuristics for extracting these principal concepts from a set of user stories.

Applying Visual Narrator to four sets of user stories produced state-of-the-art for recall and precision for detecting entities and relationships. In the best case scenario of MichiganState, the overall averages were 97% recall and 98% precision. This is possible thanks to the careful selection of heuristics with the potential of achieving the Perfect Recall Condition results, combined with our application of these heuristics based on careful dissection of user stories' syntactical properties. Especially when user stories are concise statements of the problem to solve and not lengthy descriptions of the solution, identifying the principal concepts of a user story collection is possible.

Conclusion IV

Thanks to its concise and simple structure, a user story is well-suited for automatically extracting its most important entities and relationships. Practitioners can use Visual Narrator to extract these principal concepts with state-of-the-art precision and recall, demonstrating the potential of applying well-known heuristics to user stories with natural language processing.

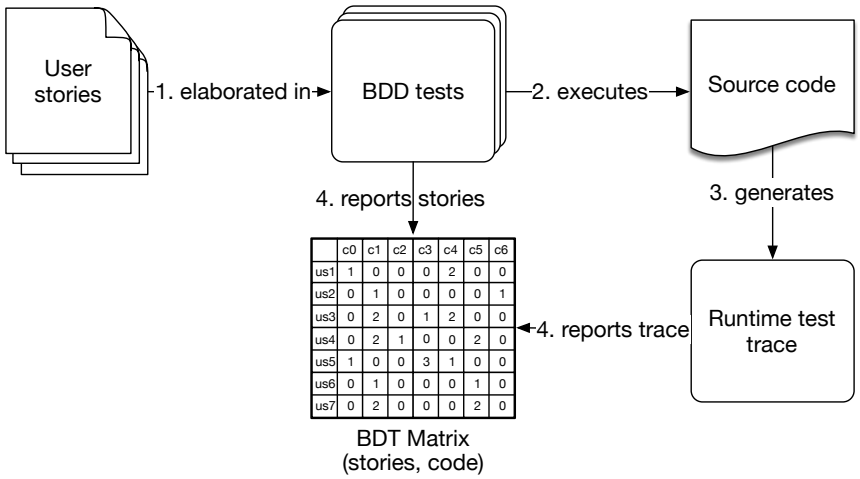
RQ5 — *How to visually organize the concepts of a user story collection?* While it is possible to extract a user story's principal concepts using natural language processing, simply presenting these textually as a list still obfuscates which entities and relationships are the most relevant. Furthermore, simple conceptual models that indiscriminately incorporate all concepts quickly become too large to be effectively explored by analysts. To filter or emphasize a specific subset of user story concepts, Chapter 6 built on Shneiderman's *visual information seeking* mantra: "overview first, zoom/filter, details on demand" (Shneiderman, 1996). We proposed and experimented with clustering mechanisms for improving the visualization of Visual Narrator's conceptual models that leverages state-of-the-art, corpus-based *semantic relatedness* algorithms based on neural networks to determine the similarity between concepts (Goldberg and Levy, 2014; Trask et al., 2015). The novelty of our approach is in its fully-automatic nature: it does not require any additional documentation about the system under development; instead, it relies on publicly available corpora of data from the Web.

Applying a prototype implementation of this approach to four real-world user story sets, we reported both positive and negative key findings. While on the one hand the generated visualizations are capable of highlighting relevant information classifications and appointing meaningful representative terms for clusters, on the other hand the output frequently includes residual clusters that cannot be assigned any label because the semantic relatedness library's vocabulary is limited, and the overview's intercluster relationships are useless because too many are visible. Nevertheless, the positive observations highlight the potential of organizing a user story collection's concepts in this way for increasing practitioners' user story understanding.

Conclusion V

By combining a user story's concepts extracted using Visual Narrator with recent advances in semantic relatedness, it is possible to produce visualizations of a user story collection at multiple levels of granularity. While the output of our prototype tool is not perfect, practitioners may explore their user story collection by filtering on and zooming in on specific entities and relationships.

RQ6 — *How to trace a user story to the source code it executes?*
Created as a part of Agile software development, a user story should lead to the creation of a piece of software. By keeping track of which source code realizes which user story, developers' program comprehension and software development speed increases. However, practitioners do not believe these benefits outweigh the effort of maintaining this trace (Arkley and Riddle, 2005). Chapter 7 presented the *Behavior-Driven Traceability method* that establishes traceability between a user story and the source code that implements it. We take advantage of automated acceptance tests created as part of the software development process Behavior-Driven Development (BDD) to construct a BDT matrix that captures the source code invoked to realize each user story.



	c0	c1	c2	c3	c4	c5	c6
us1	1	0	0	0	2	0	0
us2	0	1	0	0	0	0	1
us3	0	2	0	1	2	0	0
us4	0	2	1	0	0	2	0
us5	1	0	0	3	1	0	0
us6	0	1	0	0	0	1	0
us7	0	2	0	0	0	2	0

Figure 8.2: The Behavior Driven Traceability Method (BDT)

Figure 8.2 shows an overview of the BDT method. First, the developers *(1) elaborate* the user stories into BDD tests. Running the BDD test suite *(2) executes* the source code, which in turn *(3) generates* runtime test traces of the BDD tests. The BDT method then *(4) combines the reported* runtime test trace with the BDD test's annotated user stories into the **BDT Matrix**.

This BDT matrix contains the entire trace of a user story to the source code it executes. To reduce the noise in this output, we proposed normalized versions of the BDT matrix that emphasize the uniquely relevant methods for a specific user story. The normalized BDT matrix creates opportunities for advanced analyses such as automatically determining which source code a new user story is likely to impact. However, empirical evaluation of these analyses' effectiveness is left for future work.

Conclusion VI
Practitioners who develop software with Behavior Driven Development (BDD) may readily take advantage of the Behavior-Driven Traceability method to establish ubiquitous traceability between a user story and the source code that it executes. This demonstrates the opportunity BDD offers practitioners to overcome the traceability benefit problem.

8.1 LIMITATIONS AND THREATS TO VALIDITY

As discussed in the threats to validity section of every chapter, each of our studies had its respective limitations. A universal threat is the exclusive focus on user stories, which makes it a challenge to generalize the results to all types of requirements. Three other issues were recurring problems during our research and deserve emphasis so that the reader can determine the applicability and relevance of this work in his or her situation:

One-country bias — Despite our best efforts, the majority of collected data and participating respondents that form the empirical underpinnings of this research originated from organizations in the Netherlands. As a consequence, many of the English-written user stories we examined were created by non-native English speakers. This reduces the degree to which the reported results are generalizable to other situations and people: it is unclear whether there would be as many QUS framework criteria violations in user stories created by a native English speaker or a non-native English speaker with a different cultural background. Some cultures are more inclined to follow guidelines and express themselves in short statements, while others formulate lengthy sentences with many subordinate clauses (Taylor and Tingguag, 1991). This threat to external validity is not always problematic. In Chapter 2 the answers by Netherlands-based respondents did not statistically differentiate from other respondents. However, due to the little international empirical data in the other studies, we were unable to incorporate more of these tests. This is a recurring problem in requirements engineering research. For example, none of the papers that were awarded the Requirements Engineering Conference's Most Influential Paper Award in the past ten years included empirical data from more than one country.

Acquiescence bias — or yea-saying is the tendency of a respondent to agree with a statement, while he/she actually disagrees or is neutral. This may occur because of the way the question is formulated or because you want the interviewer to succeed. The surveys in Chapter 2 and 4 are susceptible to this threat as they included a small number of primarily positively keyed items in an attempt to reduce the non-completion rate. Furthermore, due to time constraints almost all the evaluations of the tools in Chapter 4 and 5 were conducted with participants from the authors' personal network.

Confounding — The complex and chaotic environment of a software producing organization makes it difficult to directly measure the impact of an intervention. After introducing a specific treatment to one variable you may measure a positive change in a second variable and conclude that your treatment was a success. There is, however, the risk of a confounding variable: a third variable that is the actually responsible factor for the positive change. In Chapter 4 there is the risk that simply instructing participants to pay attention to user story quality when creating them could be enough to explain the change in intrinsic user story quality after introducing QUS and AQUSA.

Perception vs. reality — the surveys and interviews conducted as part of our exploratory user story research in Chapter 2, and the evaluations of our tools in Chapters 3, 4 and 5 primarily collect data that captures the participants' *perception*. While relevant and worthwhile, this does not necessarily reflect the *actual* effectiveness of user stories or the *real* impact of introducing the Grimm User Story method. This introduces the risk that our results are too positive. Perhaps employing user stories actually has no effect at all, despite participants experiencing and reporting a beneficial effect.

8.2 FUTURE WORK

Global user story adoption shows no signs of slowing down. Slowly but surely, academic research on user stories is following suit. This dissertation contributed to this trend by exploring how to employ computational linguistic techniques to increase user story understanding. Although our work has demonstrated the potential of user stories, its exploratory nature does not always warrant making definitive claims. This is a common issue with software engineering studies that is difficult to overcome due to the chaotic environment in which we conduct our work. Nevertheless, rigorous replications of our research are necessary to confirm the findings presented in this dissertation. Furthermore, as user stories are increasingly becoming an established research domain, academia will start to be able to apply recent computer science advances to great effect such as deep learning, neural networks or conversational systems. Below we discuss three promising research lines that would contribute to fully understanding user stories.

FW1 — *Advanced Machine Learning*
The diffusion of advanced machine learning libraries, tools and knowledge is rapidly expanding. Sophisticated yet comprehensible libraries such as Google's TensorFlow combined with continuously growing computing power are enabling more and more academics to incorporate machine learning in their work. Future user story research should focus on teaching a machine to automatically identify and correct issues in a user story based on how humans previously edited and corrected their user stories. Given sufficient historic user story data, this approach may reduce the necessity for a human with understanding of the project's context to detect quality characteristic violations. A first step towards this goal is to create a binary classifier that will separate user stories with quality characteristic violations from those without any violations.

FW2 — *Linking user stories to software development artefacts*
This dissertation's contributions create opportunities for connecting user stories to other artifacts created during software development. In particular, the work on improving the quality of user stories, extracting the primary concepts of user stories, and tracing a user story to the source code that it invokes are the foundation for creating a next-generation tool in support of software engineering. We envision a tool that combines high quality user stories with popular software development tools such as GitHub, Jira and Travis CI to generate real-time reports and analyses on demand. For example, clicking on a specific role in a conceptual model displays all user stories that this role is involved in and dynamically generates a list of all developers currently working

on the user stories according to Jira, as well as a class diagram based on all the source code related to the user stories. Furthermore, Travis CI may store screenshots of all scenario steps in a project's automated acceptance tests so that users can always access an up-to-date visual representation of what a user story does. Using this tool on a touchscreen video-wall provides the development team and their managers with a central status dashboard and communication hub for monitoring and discussing a project's progress.

FW3 — *Industry adoption monitor*
Although the findings reported in Chapter 2 are a useful reflection of how practitioners work with user stories, the data only captures the situation in 2015. Replications are necessary to determine which of our results remain true over longer periods of time. Furthermore, the scope of this study was rather small. A larger-scale study with an order of magnitude more participants could explore more topics, while still posing a limited number of questions. A global, (bi-)yearly user story monitor could investigate many different topics to improve researchers' understanding of user story practice and identify promising research questions similar to the NAPIRE initiative for requirements in general by Méndez et al. (2016). Topics of interest include questions such as:

1. *How many practitioners create visualizations of their user stories for communication?*
2. *How many organizations have successfully engaged customers so that they create user stories themselves?*
3. *In what kind of projects are user stories predominantly used? Does their usage go beyond UI-based systems?*

FW4 — *Perfect Recall*
Throughout this dissertation we emphasized the importance of creating reliable tools that can extract concepts or detect defects without missing any cases: the Perfect Recall Condition. If the analyst is assured that the tool has not missed any concepts or defects, she is no longer required to manually recheck the tool's output. Although we never truly fulfilled the Perfect Recall Condition, the tools introduced in this dissertation do come close. In the case of five companies' user stories in Chapter 3 we were able to fulfill it. Nevertheless, there is still room for improvement. In future work careful gap analyses will identify opportunities to further improve the recall of the tools. The three opportunities for future work identified above will also result in new techniques and data that contribute towards this goal. Although it is currently impossible to truly achieve perfect recall, every step forward further supports our end-goal of improving user story understanding.

8.3 REFLECTIONS

Working towards a PhD is a stressful undertaking. Recent news articles reported that one third of Dutch PhD candidates experience serious psychological problems and run the risk of burn-out or depression. Thankfully, things are a bit different in our group and I am lucky to have completed this dissertation mostly unscathed. My nearest and dearest know I have my vices, yet somehow manage to remain scarily grounded. During my tenure at Utrecht University I have learned, observed and experienced many things. In this final chapter, I reflect upon three phenomena of academia that deserve to be highlighted.

Keep it simple, stupid — It seems as if every day I emphasize the importance of keeping your research simple. As an academic, it is alluring to focus on difficult problems and devise complex solutions. After all, only the most difficult problems are worthy of your brain power and therefore deserve meticulously created solutions. Difficult problems, however, often encompass so many facets that they are nearly impossible to solve. Difficult problems are difficult to understand and even more difficult to contribute towards. Too often, I read or review papers that introduce multiple major problems and then present one comprehensive solution that supposedly covers all combinations, variations and exceptions. The initial impression might be impressive, but deciphering the solution is unnecessarily tiring and challenging. Mathematicians take this to the extreme. They take decades to come up with a solution to one of their unsolvable problems, and when they finally do, it takes other mathematicians multiple years to verify whether the solution is correct or not. My research takes a completely contrary approach. Nearly all my work combines existing techniques and theories to construct a new approach that is particularly effective when applied to a hyper-specific scope. And not without merit; my work has been well received by both academia and industry. Geniuses aside, I recommend young academics to focus on narrowly-defined problems for which you can devise an elegant solution in order to generate meaningful impact.

The teaching dilemma — Education is arguably the strongest driver of economic growth. Organizations like the OECD frequently publish reports claiming that educating the general population would produce astronomical economical growth. By sharing their knowledge, universities enable the next generation to improve industry production and reap the associated benefits. Perhaps even more important, teaching is a tremendously satisfying endeavor. Throughout my PhD I have had the pleasure of teaching approximately 150 students and professionals about the world of software product management.

Yet, teaching is the most exhausting job within a university. Not only does a lecturer get to have fun preparing and delivering exciting lectures, he also needs to comply with a seemingly endless stream of bureaucracy, rules and busywork like grading exams. To make matters worse, computers, software, and software engineering practices are always changing, rendering it nearly impossible to meet your students' thirst for the latest and the greatest. Altogether, this confronts the academic with a dilemma: either complete your teaching duties imperfectly or choose to conduct research in your free time. Problematic, as research is what academics want to be doing, and are encouraged to be doing by aforementioned academic treadmill. On the other hand, demand for software engineering teaching staff has never been higher and is likely to continue given the growing number of students interested in the world of computers. Confronted by these perspectives, should the young academic focus on improving his teaching skills or his research skills?

The academic treadmill — "Publish or perish" accurately characterizes the current academic climate. Academics experience pressure to publish as much as possible, while publication venues strive to strictly publish novel research that might result in a newspaper headline. Together, these forces create the incentive to continuously work on new initiatives and ideas at the expense of existing but incomplete projects. The result? Densely populated academic graveyards with projects that no one contributes to anymore. Our work is no exception, the last time I updated the AQUSA GitHub project was on August 12, 2016 and, as long as there are no substantial advances in linguistic tooling, there is little incentive for me to pick this up again. Apparently, there is no escaping the allure of new and exciting projects.

Bibliography

Z. S. H. Abad, G. Ruhe, and M. Noaeen. Requirements Engineering Visualization: A Systematic Literature Review. In *Proceedings of the International Requirements Engineering Conference (RE)*. IEEE, 2016. (Cited on page 140.)

S. Abrahão, E. Insfran, J. A. Carsí, and M. Genero. Evaluating requirements modeling methods based on user perceptions: A family of experiments. *Information Sciences*, 181(16):3356–3378, 2011. ISSN 0020-0255. (Cited on pages 24 and 40.)

G. Aguado De Cea, A. Gómez-Pérez, E. Montiel-Ponsoda, and M. C. Suárez-Figueroa. Natural Language-Based Approach for Helping in the Reuse of Ontology Design Patterns. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, volume 5268 of *LNCS*, pages 32–47. Springer, 2008. (Cited on page 110.)

J. Akoka and I. Comyn-Wattiau. Entity-Relationship and Object-Oriented Model Automatic Clustering. *Data & Knowledge Engineering*, 20(2):87–117, 1996. (Cited on page 157.)

V. Ambriola and V. Gervasi. On the Systematic Analysis of Natural Language Requirements with CIRCE. *Automated Software Engineering*, 13(1):107–167, 2006. (Cited on pages 138 and 158.)

J. Aranda, N. Ernst, J. Horkoff, and S. Easterbrook. A Framework for Empirical Evaluation of Model Comprehensibility. In *Proceedings of the Workshop on Modelling in Software Engineering (MiSE)*, 2007. (Cited on pages 11, 15, 107, and 146.)

B. Arendse and G. Lucassen. Toward tool mashups: Comparing and combining nlp re tools. In *Proceedings of the International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pages 26–31, 2016. (Cited on page 138.)

P. Arkley and S. Riddle. Overcoming the traceability benefit problem. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 385–389, 2005. (Cited on pages 11, 15, 162, and 193.)

C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer. Improving Requirements Glossary Construction via Clustering: Approach and Industrial Case Studies. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2014. ISBN 978-1-4503-2774-9. (Cited on page 79.)

C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer. Extracting Domain Models from Natural-language Requirements: Approach and Industrial Evaluation. In *Proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 250–260. ACM, 2016. (Cited on page 139.)

R. Barbosa, A. E. A. Silva, and R. Moraes. Use of Similarity Measure to Suggest the Existence of Duplicate User Stories in the Scrum Process. In *Proceedings of the Annual International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 2–5, 2016. (Cited on pages 136 and 137.)

K. Beck. Embracing change with extreme programming. *Computer*, 32(10): 70–77, October 1999. (Cited on pages 2, 13, 44, and 45.)

K. Beck. *Test-driven development: By example*. Addison-Wesley, 2003. (Cited on page 165.)

K. Beck and M. Fowler. *Planning Extreme Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 2000. ISBN 0201710919. (Cited on page 44.)

K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for agile software development, 2001. URL http://www.agilemanifesto.org/. (Cited on pages 2 and 6.)

K. Bedell. Opinions on opinionated software. *Linux Journal*, 2006(147):1–, July 2006. ISSN 1075-3583. (Cited on page 31.)

D. Berry, R. Gacitua, P. Sawyer, and S. Tjong. The Case for Dumb Requirements Engineering Tools. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 7195 of *LNCS*, pages 211–217. Springer, 2012. (Cited on pages 7, 44, 45, 58, 78, 79, 80, and 137.)

D. M. Berry and E. Kamsties. Ambiguity in Requirements Specification. In *Perspectives on Software Requirements*, volume 753 of *International Series in Engineering and Computer Science*, pages 7–44. Springer, 2004. ISBN 978-1-4613-5090-3. (Cited on page 53.)

D. M. Berry, E. Kamsties, and M. M. Krieger. From Contract Drafting to Software Specification: Linguistic Sources of Ambiguity. Technical report, School of Computer Science, University of Waterloo, ON, Canada, 2001. (Cited on pages 9 and 102.)

D. M. Berry, K. Czarnecki, M. Antkiewicz, and M. Abdelrazik. The problem of the lack of benefit of a document to its producer (PotLoBoaDtiP). In *Proceedings of the International Conference on Software Science, Technology and Engineering (ICSTE)*, pages 37–42, 2016. (Cited on page 162.)

N. Bik, G. Lucassen, and S. Brinkkemper. A reference method for user story requirements in agile systems development. In *Proceedings of the International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 292–298, 2017.

F. Blaauboer, K. Sikkel, and M. N. Aydin. Deciding to adopt requirements traceability in practice. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 294–308, 2007. ISBN 978-3-540-72988-4. (Cited on pages 11 and 162.)

B. W. Boehm. Understanding and controlling software costs. *Journal of Parametrics*, 8(1):32–68, 1988. (Cited on page 31.)

H. N. Boone and D. A. Boone. Analyzing Likert data. *Journal of Extension*, 50(2):1–5, 2012. (Cited on pages 18 and 30.)

M. Borg, P. Runeson, and A. Ardö. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering*, 19(6):1565–1616, 2014. ISSN 1573-7616. (Cited on pages 12 and 162.)

E. Bouillon, P. Mäder, and I. Philippow. A survey on usage scenarios for requirements traceability in practice. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 158–173, 2013. ISBN 978-3-642-37422-7. (Cited on pages 11, 15, and 162.)

E. S. Btoush and M. M. Hammad. Generating ER Diagrams from Requirement Specifications Based on Natural Language Processing. *International Journal of Database Theory and Application*, 8(2):61–70, 2015. (Cited on page 111.)

A. Bucchiarone, S. Gnesi, and P. Pierini. Quality Analysis of NL Requirements: An Industrial Case Study. In *Proceedings of the International Conference on Requirements Engineering (RE)*, pages 390–394, 2005. (Cited on page 44.)

L. Cao and B. Ramesh. Agile requirements engineering practices: An empirical study. *IEEE Software*, 25(1):60–67, 2008. (Cited on pages 13, 44, and 164.)

J. G. Carbonell. Politics: Automated ideological reasoning*. *Cognitive Science*, 2(1):27–51, 1978. ISSN 1551-6709. (Cited on page 6.)

J. S. Chall and E. Dale. *Readability revisited: The new Dale-Chall readability formula*. Brookline Books, 1995. (Cited on page 89.)

P. P. Chen. Entity-Relationship Diagrams and English Sentence Structure. In *Proceedings of the International Conference on the Entity-Relationship Approach to Systems Analysis and Design*, pages 13–14, 1983. (Cited on pages 110 and 111.)

B. H. C. Cheng and J. M. Atlee. Research directions in requirements engineering. In *Proceedings of Future of Software Engineering (FOSE)*, FOSE '07, pages 285–303, Washington, DC, USA, 2007. IEEE. ISBN 0-7695-2829-5. (Cited on page 5.)

E. H.-H. Chi. A Taxonomy of Visualization Techniques using the Data State Reference Model. In *Proceedings of the Information Visualization Conference (InfoVis)*, pages 69–75. IEEE, 2000. (Cited on page 140.)

J. Choma, L. A. M. Zaina, and D. Beraldo. *UserX Story: Incorporating UX Aspects into User Stories Elaboration*, pages 131–140. Springer, 2016. (Cited on page 136.)

A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. Validation of Requirements for Hybrid Systems: A Formal Approach. *ACM Transactions on Software Engineering and Methodology*, 21(4):22:1–22:34, Feb. 2013. (Cited on page 78.)

D. L. Clason and T. J. Dormody. Analyzing data measured by individual Likert-type items. *Journal of Agricultural Education*, 35(4):31–35, 1994. (Cited on page 30.)

J. Cleland-Huang. Traceability in Agile Projects. In *Software and Systems Traceability*, pages 265–275. Springer, 2012. (Cited on pages 141 and 164.)

J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova. Best Practices for Automated Traceability. *Computer*, 40(6):27–35, 2007. (Cited on page 44.)

J. Cleland-Huang, O. C. Z. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman. Software traceability: Trends and future directions. In *Proceedings of Future of Software Engineering (FOSE)*, pages 55–69. ACM, 2014a. ISBN 978-1-4503-2865-4. (Cited on pages 11, 162, 164, and 184.)

J. Cleland-Huang, M. Rahimi, and P. Mäder. Achieving lightweight trustworthy traceability. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE)*, pages 849–852, 2014b. (Cited on page 164.)

M. Cohn. *User Stories Applied: for Agile Software Development.* Addison Wesley Professional, Redwood City, CA, USA, 2004. (Cited on pages 2, 6, 13, 24, 27, 28, 29, 44, 45, 46, 48, 86, 87, 99, 104, 105, 108, 111, 129, 137, 141, and 164.)

N. Condori-Fernandez, M. Daneva, K. Sikkel, R. Wieringa, O. Dieste, and O. Pastor. A systematic mapping study on empirical evaluation of software requirements specifications techniques. In *Proceedings of the International Symposium on Empirical Software Engineering and Management (ESEM)*, pages 502–505. IEEE, 2009. ISBN 978-1-4244-4842-5. (Cited on pages 24 and 40.)

A. Cooper. *The Inmates Are Running the Asylum.* Macmillan Publishing Co., Inc., Indianapolis, IN, USA, 1999. ISBN 0672316498. (Cited on page 47.)

J. R. Cooper Jr, S.-W. Lee, R. A. Gandhi, and O. Gotel. Requirements Engineering Visualization: A Survey on the State-of-the-art. In *Proceedings of the International Workshop on Requirements Engineering Visualization (REV)*, pages 46–55, 2009. (Cited on pages 140 and 156.)

K. Cox, M. Niazi, and J. Verner. Empirical study of sommerville and sawyer's requirements engineering practices. *IET Software*, 3(5):339–355, 2009. ISSN 1751-8806. (Cited on page 5.)

B. Craft and P. Cairns. Beyond Guidelines: What Can We Learn from the Visual Information Seeking Mantra? In *Proceedings of the International*

Conference Information Visualization (IV), pages 110–118, 2005. (Cited on pages 148, 151, and 152.)

J. Cruz-Lemus, M. Genero, S. Morasca, and M. Piattini. Using practitioners for assessing the understandability of uml statechart diagrams with composite states. In *Advances in conceptual modeling – foundations and applications*, volume 4802 of *LNCS*, pages 213–222. Springer, 2007. ISBN 978-3-540-76291-1. (Cited on pages 24 and 40.)

F. Dalpiaz, X. Franch, and J. Horkoff. istar 2.0 language guide. *CoRR*, abs/1605.07767, 2016. URL http://arxiv.org/abs/1605.07767. (Cited on page 137.)

D. Damian, J. Chisan, L. Vaidyanathasamy, and Y. Pal. Requirements engineering and downstream software development: Findings from a case study. *Empirical Software Engineering*, 10(3):255–283, 2005. ISSN 1573-7616. (Cited on pages 97 and 98.)

I. Davies, P. Green, M. Rosemann, M. Indulska, and S. Gallo. How do Practitioners Use Conceptual Modeling in Practice? *Data & Knowledge Engineering*, 58(3):358–380, 2006. (Cited on pages 9, 10, 15, and 102.)

A. M. Davis. *Software Requirements: Objects, Functions, and States*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. ISBN 0-13-805763-X. (Cited on page 8.)

C. W. H. Davis. *Agile metrics in action: Measuring and enhancing the performance of agile teams*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2015. ISBN 1617292486, 9781617292484. (Cited on page 86.)

R. A. de Carvalho, F. L. de Carvalho e Silva, and R. S. Manhães. Mapping business process modeling constructs to behavior driven development ubiquitous language. *CoRR*, abs/1006.4892, 2010. (Cited on page 168.)

O. Dieste and N. Juristo. Systematic review and aggregation of empirical studies on elicitation techniques. *Transactions on Software Engineering*, 37 (2):283–304, 2011. ISSN 0098-5589. (Cited on pages 24 and 40.)

S. Dimitrijević, J. Jovanović, and V. Devedžić. A Comparative Study of Software Tools for User Story Management. *Information and Software Technology*, 57:352–368, 2015. (Cited on pages 76 and 137.)

B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95, 2013. ISSN 2047-7481. (Cited on page 172.)

S. Du and D. P. Metzler. An Automated Multi-component Approach to Extracting Entity Relationships from Database Requirement Specification Documents. In *Proceedings of the International Conference on Applications of Natural Language to Information Systems (NLDB)*, volume 3999 of *LNCS*, pages 1–11. Springer, 2006. (Cited on pages 10, 102, 141, and 146.)

A. M. D. Duarte, D. Duarte, and M. Thiry. TraceBoK: Toward a software requirements traceability body of knowledge. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 236–245, 2016. (Cited on pages 12 and 162.)

M. Dudáš, O. Zamazal, and V. Svátek. Roadmapping and navigating in the ontology visualization landscape. In K. Janowicz, S. Schlobach, P. Lambrix, and E. Hyvönen, editors, *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 137–152. Springer, 2014. (Cited on page 140.)

J.-F. Dumas-Monette and S. Trudel. Requirements Engineering Quality Revealed through Functional Size Measurement: An Empirical Study in an Agile Context. In *Proceedings of the International Workshop on Software Measurement (IWSM)*, pages 222–232, 2014. (Cited on page 77.)

K. Ellis and D. M. Berry. Quantifying the impact of requirements definition and management process maturity on project outcome in large business application development. *Requirements Engineering*, 18(3):223–249, 2013. ISSN 1432-010X. (Cited on page 5.)

N. A. Ernst and G. C. Murphy. Case studies in just-in-time requirements analysis. In *Proceedings of the International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 25–32, 2012. (Cited on pages 162 and 165.)

F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. An automatic quality evaluation for natural language requirements. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 1, pages 4–5, 2001. (Cited on page 9.)

S. Farfeleder, T. Moser, A. Krall, T. Stålhane, H. Zojer, and C. Panis. DODT: Increasing Requirements Formalism using Domain Ontologies for Improved

Embedded Systems Development. In *Proceedings of the International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 271–274, 2011. (Cited on page 78.)

W. Farid and F. Mitropoulos. Visualization and Scheduling of Non-Functional Requirements for Agile Processes. In *Proceedings of the Region 3 Technical, Professional, and Student Conference (Southeastcon)*, pages 1–8, 2013. (Cited on page 77.)

H. Femmer, D. M. Fernández, E. Juergens, M. Klose, I. Zimmer, and J. Zimmer. Rapid Requirements Checks with Requirements Smells: Two Case Studies. In *Proceedings of the International Workshop on Rapid Continuous Software Engineering (RCoSE)*, pages 10–19, 2014. (Cited on page 78.)

P. Fettke. How Conceptual Modeling is Used. *Communications of the Association for Information Systems*, 25(1):43, 2009. (Cited on pages 9 and 102.)

M. D. Fraser, K. Kumar, and V. K. Vaishnavi. Strategies for incorporating formal specifications in software development. *Communications of the ACM*, 37(10):74–86, Oct. 1994. ISSN 0001-0782. (Cited on page 8.)

F. Friedrich, J. Mendling, and F. Puhlmann. Process Model Generation from Natural Language Text. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 6741 of *LNCS*, pages 482–496. Springer, 2011. (Cited on page 79.)

R. Gacitua, P. Sawyer, and V. Gervasi. On the Effectiveness of Abstraction Identification in Requirements Engineering. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 5–14, 2010. (Cited on page 45.)

C. L. Gagné. Lexical and Relational Influences on the Processing of Novel Compounds. *Brain and Language*, 81(1–3):723–735, 2002. (Cited on page 113.)

R. A. Gandhi and S.-W. Lee. Discovering and understanding multi-dimensional correlations among certification requirements with application to risk assessment. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 231–240, 2007. (Cited on page 156.)

N. Gao and Z. Li. Generating testing codes for behavior-driven development from problem diagrams: A tool-based approach. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 399–400, 2016. (Cited on page 168.)

N. Genon, P. Heymans, and D. Amyot. Analysing the Cognitive Effectiveness of the BPMN 2.0 Visual Notation. In *Proceedings of the ACM SIGPLAN International Conference on Software Language Engineering (SLE)*, pages 377–396. Springer, 2010. (Cited on pages 139 and 140.)

G. Génova, J. M. Fuentes, J. Llorens, O. Hurtado, and V. Moreno. A framework to measure and improve the quality of textual requirements. *Requirements Engineering*, 18(1):25–41, 2013. ISSN 1432-010X. (Cited on page 9.)

M. Glinz. Improving the Quality of Requirements with Scenarios. In *Proceedings of the World Congress on Software Quality (WCSQ)*, pages 55–60, 2000. (Cited on pages 8, 49, and 77.)

V. Goel. *Sketches of Thought: A Study of the Role of Sketching in Design Problem-solving and Its Implications for the Computational Theory of the Mind*. PhD thesis, Berkeley, CA, USA, 1991. UMI Order No. GAX92-28664. (Cited on page 8.)

Y. Goldberg and O. Levy. word2vec Explained: Deriving Mikolov et al.'s Negative-sampling Word-embedding Method. *arXiv preprint arXiv:1402.3722*, 2014. (Cited on pages 11, 107, 146, and 192.)

A. Gomez, G. Rueda, and P. Alarcón. A Systematic and Lightweight Method to Identify Dependencies between User Stories. In *Proceedings of the International Conference on Agile Software Development (XP)*, volume 48 of *LNBIP*, pages 190–195. Springer, 2010. (Cited on pages 39 and 77.)

J. Gulden and H. A. Reijers. Toward Advanced Visualization Techniques for Conceptual Modeling. In *Proceedings of the Forum at the International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 33–40, 2015. (Cited on page 139.)

R. Gunning. *Technique of clear writing*. McGraw-Hill, 1968. (Cited on page 89.)

W.-M. Han and S.-J. Huang. An empirical analysis of risk components and performance on software projects. *Journal of Systems and Software*, 80(1): 42 – 50, 2007. ISSN 0164-1212. (Cited on page 5.)

S. Harispe, S. Ranwez, S. Janaqi, and J. Montmain. *Semantic Similarity from Natural Language and Ontology Analysis*. Morgan & Claypool Publishers, 2015. (Cited on page 145.)

H. Harmain and R. Gaizauskas. CM-Builder: A Natural Language-Based CASE Tool for Object-Oriented Analysis. *Automated Software Engineering*, 10(2):157–181, 2003. (Cited on pages 102, 110, 138, 141, 146, and 158.)

S. Hartmann and S. Link. English Sentence Structures and EER Modeling. In *Proceedings of the Asia-Pacific Conference on Conceptual Modelling (APCCM)*, pages 27–35, 2007. (Cited on pages 102, 110, 111, 113, and 141.)

P. Heck and A. Zaidman. A Quality Framework for Agile Requirements: A Practitioner's Perspective. *CoRR*, 2014. (Cited on pages 9, 14, 44, 49, and 50.)

A. R. Hevner and S. Chatterjee. Design Science Research Frameworks. In *Design Research in Information Systems*, pages 23–31. Springer, New York, NY, USA, 2010. (Cited on page 16.)

R. Hoda, P. Kruchten, J. Noble, and S. Marshall. Agility in context. In *Proceedings of the International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 74–88, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0203-6. (Cited on page 24.)

H. Hofmann and F. Lehner. Requirements engineering as a success factor in software projects. *Software, IEEE*, 18(4):58–66, 2001. ISSN 0740-7459. (Cited on pages 24 and 40.)

H. Holbrook, III. A scenario-based methodology for conducting requirements elicitation. *SIGSOFT Software Engineering Notes*, 15(1):95–104, 1990. (Cited on pages 75 and 136.)

H. Holm, T. Sommestad, and J. Bengtsson. Requirements engineering: The quest for the dependent variable. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 16–25, 2015. (Cited on pages 85 and 98.)

E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. Springer-Verlag New York, Inc., New York, NY, USA, 3rd edition, 2010. ISBN 1849964041, 9781849964043. (Cited on page 13.)

IEEE. IEEE Recommended Practice for Software Requirements Specifications. *IEEE Std 830-1993*, 1994. (Cited on page 49.)

IEEE. Systems and software engineering – vocabulary. *ISO/IEC/IEEE 24765:2010(E)*, pages 1–418, 2010. (Cited on page 1.)

IEEE. Systems and software engineering – life cycle processes – requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*, pages 1–94, 2011. (Cited on pages 1, 4, 8, 14, 77, and 78.)

C. IEEE, P. Bourque, and R. E. Fairley. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE, Los Alamitos, CA, USA, 3rd edition, 2014. ISBN 0769551661, 9780769551661. (Cited on page 1.)

M. Imaz and C. Benyon. How Stories Capture Interactions. In *Proceedings of the IFIP International Conference on Human-Computer Interaction (INTERACT)*, pages 321–328, 1999. (Cited on pages 75 and 136.)

R. Jeffries. Essential xp: Card, conversation, and confirmation, August . URL https://ronjeffries.com/xprog/articles/expcardconversationconfirmation/. (Cited on pages 13 and 39.)

R. E. Jeffries, A. Anderson, and C. Hendrickson. *Extreme Programming Installed*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000. ISBN 0201708426. (Cited on page 44.)

M. Jørgensen. A survey on the characteristics of projects with success in delivering client benefits. *Information and Software Technology*, 78:83 – 94, 2016. ISSN 0950-5849. (Cited on page 5.)

M. Kamalrudin, J. Hosking, and J. Grundy. Improving Requirements Quality Using Essential Use Case Interaction Patterns. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 531–540, 2011. (Cited on page 78.)

M. I. Kamata and T. Tamai. How does requirements quality relate to project success or failure? In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 69–78, 2007. (Cited on pages 8, 77, and 97.)

M. Kassab. The Changing Landscape of Requirements Engineering Practices over the Past Decade. In *Proceedings of the International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 1–8, 2015. (Cited on pages 2, 3, 8, 24, 39, 44, 75, 82, 102, 136, 146, and 164.)

M. Kassab, C. Neill, and P. Laplante. State of Practice in Requirements Engineering: Contemporary Data. *Innovations in Systems and Software Engineering*, 10(4):235–241, 2014. (Cited on pages 102, 103, 105, 140, and 146.)

E. Knauss, C. El Boustani, and T. Flohr. Investigating the impact of software requirements specification quality on project success. In *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES)*, volume 32 of *LNBIP*, pages 28–42. Springer, 2009. ISBN 978-3-642-02152-7. (Cited on page 97.)

Kniberg, Henrik. What is Crisp? http://blog.crisp.se/2010/05/08/henrikkniberg/what-is-crisp, 2010. Accessed: 2016-05-25. (Cited on page 85.)

G. Kotonya and I. Sommerville. *Requirements Engineering: Processes and Techniques*. Wiley Publishing, 1st edition, 1998. ISBN 0471972088, 9780471972082. (Cited on page 1.)

J. Kruger and D. Dunning. Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of Personality and Social Psychology*, 77(6):1121, 1999. (Cited on page 28.)

M. Lapata. The Disambiguation of Nominalizations. *Computational Linguistics*, 28(3):357–388, 2002. (Cited on page 113.)

C. Lee, L. Guadagno, and X. Jia. An agile approach to capturing requirements and traceability. In *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 2003. (Cited on page 164.)

J. Lee and K.-Y. Lai. What's in design rationale? *Human Computer Interaction*, 6(3):251–280, 1991. ISSN 0737-0024. (Cited on pages 4 and 42.)

W. Li, X. Zhang, C. Niu, Y. Jiang, and R. Srihari. An Expert Lexicon Approach to Identifying English Phrasal Verbs. In *Proceedings of the Meeting of the Association for Computational Linguistics (ACL)*, pages 513–520, 2003. (Cited on page 114.)

J. Lin, H. Yu, Z. Shen, and C. Miao. Using Goal Net to Model User Stories in Agile Software Development. In *Proceedings of the IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence,*

Networking and Parallel/Distributed Computing (SNPD), pages 1–6. IEEE, 2014. (Cited on pages 40 and 76.)

O. I. Lindland, G. Sindre, and A. Sølvberg. Understanding Quality in Conceptual Modeling. *IEEE Software*, 11(2):42–49, 1994. (Cited on page 50.)

O. Liskin, R. Pham, S. Kiesling, and K. Schneider. Why We Need a Granularity Concept for User Stories. In *Proceedings of the International Conference on Agile Software Development (XP)*, volume 179 of *LNBIP*, pages 110–125. Springer, 2014. (Cited on pages 39, 51, and 76.)

S. Lohmann, V. Link, E. Marbach, and S. Negru. WebVOWL: Web-based Visualization of Ontologies. In *Proceedings of EKAW Satellite Events*, volume 8982 of *LNAI*, pages 154–158. Springer, 2015. (Cited on pages 10 and 107.)

P. Lombriser, F. Dalpiaz, G. Lucassen, and S. Brinkkemper. Gamified requirements engineering: Model and experimentation. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 171–187, 2016. (Cited on pages 38 and 49.)

P. Loucopoulos and V. Karakostas. *System Requirements Engineering*. McGraw-Hill, Inc., New York, NY, USA, 1995. ISBN 0077078438. (Cited on page 75.)

D. Lübke and T. van Lessen. Modeling test cases in bpmn for behavior-driven development. *IEEE Software*, 33(5):15–21, 2016. ISSN 0740-7459. (Cited on page 168.)

G. Lucassen. Materials of survey and interviews on user story practice, 2015. URL http://staff.science.uu.nl/~lucas001/user_story_materials.zip. (Cited on pages 26 and 38.)

G. Lucassen. Experimental materials QUS and AQUSA evaluation. http://www.staff.science.uu.nl/~lucas001/qus_aqusa_eval_materials.zip, 2016. Accessed: 2016-10-02. (Cited on pages 83, 84, 85, and 94.)

G. Lucassen and S. Jansen. Gamification in consumer marketing - future or fallacy? *Procedia - Social and Behavioral Sciences*, 148:194 – 202, 2014. International Conference on Strategic Innovative Marketing (ICSIM).

G. Lucassen, M. van de Keuken, F. Dalpiaz, S. Brinkkemper, G. W. Sloof, and J. Schlingmann. Jobs-to-be-done oriented requirements engineering: A method for defining job stories. In Submission.

G. Lucassen, S. Brinkkemper, S. Jansen, and E. Handoyo. Comparison of visual business modeling techniques for software companies. In *Proceedings of the International Conference on Software Business (ICSOB)*, pages 79–93, 2012.

G. Lucassen, K. van Rooij, and S. Jansen. Ecosystem health of cloud paas providers. In *Proceedings of the International Conference on Software Business (ICSOB)*, pages 183–194, 2013.

G. Lucassen, J. M. E. M. van der Werf, and S. Brinkkemper. Alignment of software product management and software architecture with discussion models. In *Proceedings of the International Workshop on Software Product Management (IWSPM)*, pages 21–30, 2014.

G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Forging High-Quality User Stories: Towards a Discipline for Agile Requirements. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 126–135, 2015. (Cited on pages 20, 24, 33, 39, 42, 44, 45, 48, 65, and 103.)

G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. The Use and Effectiveness of User Stories in Practice. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 205–222, 2016a. (Cited on pages 2, 14, 19, 44, 82, 85, 86, 94, 102, 103, 105, 144, 146, and 164.)

G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Improving Agile Requirements: The Quality User Story Framework and Tool. *Requirements Engineering*, 21(3):383–403, 2016b. (Cited on pages 20, 82, 83, 85, 103, 104, 105, 106, 114, 117, 123, 127, 134, 135, 138, and 144.)

G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Visualizing user story requirements at multiple granularity levels via semantic relatedness. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, pages 463–478, 2016c. (Cited on pages 21, 103, 105, 107, 137, and 140.)

G. Lucassen, F. Dalpiaz, J. M. E. M. v. d. Werf, S. Brinkkemper, and D. Zowghi. Behavior-driven requirements traceability via automated

acceptance tests. In *Proceedings of the International Workshop on Just-In-Time Requirements Engineering (JIT-RE)*, pages 431–434, 2017a.

G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Improving user story practice with the grimm method: A multiple case study in the software industry. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 235–252, 2017b. (Cited on page 20.)

G. Lucassen, M. Robeer, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Extracting conceptual models from user stories with visual narrator. *Requirements Engineering*, 22(3):339–358, 2017c. (Cited on page 21.)

A. Lucia and A. Qusef. Requirements Engineering in Agile Software Development. *Journal of Emerging Technologies in Web Intelligence*, 2(3), 2010. (Cited on page 77.)

P. Mäder and A. Egyed. Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering*, 20(2):413–441, 2015. ISSN 1573-7616. (Cited on pages 11, 15, and 162.)

L. Madeyski and M. Jureczko. Which process metrics can significantly improve defect prediction models? An empirical study. *Software Quality Journal*, 23 (3):393–422, 2015. ISSN 1573-1367. (Cited on page 86.)

V. Mahnič and T. Hovelja. On Using Planning Poker for Estimating User Stories. *Journal of Systems and Software*, 85(9):2086–2095, 2012. (Cited on pages 13 and 76.)

C. D. Manning. *Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?*, pages 171–189. Springer, Berlin, Heidelberg, 2011. ISBN 978-3-642-19400-9. (Cited on page 134.)

C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, USA, 1999. (Cited on page 6.)

C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*. Cambridge university press Cambridge, 2008. (Cited on pages 122 and 123.)

S. T. March and G. F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251 – 266, 1995. ISSN 0167-9236. (Cited on page 16.)

T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976. ISSN 0098-5589. (Cited on page 85.)

S. McConnell. An ounce of prevention. *Software*, 18(3):5–7, 2001. ISSN 0740-7459. (Cited on page 31.)

N. R. Mead. A history of the international requirements engineering conference (re). In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 21–221, 2013. (Cited on page 4.)

D. F. Méndez, S. Wagner, M. Kalinowski, M. Felderer, P. Mafra, A. Vetrò, T. Conte, M.-T. Christiansson, D. Greer, C. Lassenius, T. Männistö, M. Nayabi, M. Oivo, B. Penzenstadler, D. Pfahl, R. Prikladnicki, G. Ruhe, A. Schekelmann, S. Sen, R. Spinola, A. Tuzcu, J. L. de la Vara, and R. Wieringa. Naming the pain in requirements engineering. *Empirical Software Engineering*, pages 1–41, 2016. ISSN 1573-7616. (Cited on pages 5 and 198.)

D. Méndez Fernández and S. Wagner. A case study on artefact-based RE improvement in practice. In *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES)*, volume 9459 of *LNCS*, pages 114–130. Springer, 2015. ISBN 978-3-319-26844-6. (Cited on pages 98 and 100.)

T. Merten, D. Krämer, B. Mager, P. Schell, S. Bürsner, and B. Paech. Do information retrieval algorithms for automated traceability perform effectively on issue tracking system data? In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 45–62, 2016. ISBN 978-3-319-30282-9. (Cited on pages 12 and 162.)

R. Mesquita, A. Jaqueira, C. Agra, M. Lucena, and F. Alencar. US2StarTool: Generating *i** Models from User Stories. In *Proceedings of the International i* Workshop (iStar)*, 2015. (Cited on pages 40, 76, 137, and 139.)

F. Meziane and S. Vadera. Obtaining E-R Diagrams Semi-Automatically from Natural Language Specifications. In *Proceedings of the International Conference on Enterprise Information Systems (ICEIS)*, pages 638–642, 2004. (Cited on pages 110 and 111.)

L. Mich. NL-OOPS: From Natural Language to Object Oriented Requirements Using the Natural Language Processing System LOLITA. *Natural Language Engineering*, 2:161–187, 6 1996. ISSN 1469-8110. (Cited on pages 138 and 158.)

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. 2013. (Cited on pages 6 and 145.)

G. A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995. (Cited on page 65.)

E. Miranda, P. Bourque, and A. Abran. Sizing User Stories using Paired Comparisons. *Information and Software Technology*, 51(9):1327–1337, 2009. (Cited on pages 13 and 76.)

R. Mohanani, P. Ralph, and B. Shreeve. Requirements fixation. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 895–906, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2756-5. (Cited on page 5.)

D. Moody. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering*, 35(6):756–779, 2009. (Cited on pages 11, 15, 107, 139, and 146.)

D. Moody and A. Flitman. A Methodology for Clustering Entity Relationship Models—A Human Information Processing Approach. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, pages 114–130. 1999. (Cited on pages 149, 151, and 157.)

D. L. Moody, P. Heymans, and R. Matulevičius. Visual Syntax does Matter: Improving the Cognitive Effectiveness of the i* Visual Notation. *Requirements Engineering*, 15(2):141–175, 2010. (Cited on page 140.)

N. P. Napier, L. Mathiassen, and R. D. Johnson. Combining perceptions and prescriptions in requirements engineering process assessment: An industrial case study. *IEEE Transactions on Software Engineering*, 35(5):593–606, 2009. ISSN 0098-5589. (Cited on page 98.)

C. J. Neill and P. A. Laplante. Requirements Engineering: The State of the Practice. *IEEE Software*, 20(6):40, 2003. (Cited on page 102.)

D. North. Behavior modification, June 2006. URL https://www.stickyminds.com/better-software-magazine/behavior-modification. (Cited on pages 12, 162, 165, and 166.)

B. Nuseibeh and S. Easterbrook. Requirements engineering: A roadmap. In *Proceedings of Future of Software Engineering (FOSE)*, ICSE '00, pages 35–46, New York, NY, USA, 2000. ACM. ISBN 1-58113-253-0. (Cited on page 4.)

N. Omar, J. Hanna, and P. McKevitt. Heuristics-Based Entity-Relationship Modelling through Natural Language Processing. In *Proceedings of the Irish Conference on Artificial Intelligence & Cognitive Science (AICS)*, pages 302–313, 2004. (Cited on pages 102, 110, 111, 112, 140, and 146.)

S. P. Overmyer, B. Lavoie, and O. Rambow. Conceptual Modeling through Linguistic Analysis Using LIDA. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 401–410. IEEE, 2001. (Cited on page 102.)

S. O'Brien. Controlling controlled english. an analysis of several controlled language rule sets. pages 105–114, 2003. (Cited on page 7.)

F. Paetsch, A. Eberlein, and F. Maurer. Requirements engineering and agile software development. In *Proceedings of the International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE)*, pages 308–313, 2003. (Cited on page 13.)

E. Paja, F. Dalpiaz, and P. Giorgini. Managing Security Requirements Conflicts in Socio-Technical Systems. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, volume 8217 of *LNCS*, pages 270–283, 2013. (Cited on page 55.)

C. Patel and M. Ramachandran. Story Card Maturity Model (SMM): A Process Improvement Framework for Agile Requirements Engineering Practices. *Journal of Software*, 4(5), 2009. (Cited on page 77.)

J. Patton and P. Economy. *User story mapping: discover the whole story, build the right product*. " O'Reilly Media, Inc.", 2014. (Cited on page 13.)

V. Pekar, M. Felderer, and R. Breu. Improvement methods for software requirement specifications: A mapping study. In *International Conference on the Quality of Information and Communications Technology*, pages 242–245, 2014. (Cited on page 5.)

J. Pennington, R. Socher, and C. D. Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of EMNLP*, pages 1532–1543, 2014. (Cited on pages 6 and 145.)

B. Penzenstadler, J. Eckhardt, and D. Mendez Fernandez. Two replication studies for evaluating artefact models in RE: Results and lessons learnt. In *Proceedings of the International Workshop on Replication in Empirical Software Engineering Research (RESER)*, pages 66–75, 2013. (Cited on pages 24 and 40.)

E. J. Philippo, W. Heijstek, B. Kruiswijk, M. R. V. Chaudron, and D. M. Berry. Requirement ambiguity not as important as expected — results of an empirical evaluation. In J. Doerr and A. L. Opdahl, editors, *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 65–79, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-37422-7. (Cited on page 5.)

K. Pohl. *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer, 1st edition, 2010. ISBN 3642125778, 9783642125775. (Cited on pages 4 and 13.)

D. Popescu, S. Rugaber, N. Medvidovic, and D. M. Berry. Reducing Ambiguities in Requirements Specifications Via Automatically Created Object-Oriented Models. In *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*, volume 5320 of *LNCS*, pages 103–124. Springer, 2008a. (Cited on page 44.)

D. Popescu, S. Rugaber, N. Medvidovic, and D. M. Berry. Reducing Ambiguities in Requirements Specifications Via Automatically Created Object-Oriented Models. In *Innovations for Requirement Analysis. From Stakeholders' Needs to Formal Designs*, volume 5320 of *LNCS*, pages 103–124. Springer, 2008b. (Cited on page 108.)

C. Potts and G. Bruns. Recording the reasons for design decisions. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 418–427. IEEE, 1988. ISBN 0-89791-258-6. (Cited on pages 4 and 42.)

M. Rahimi, W. Goss, and J. Cleland-Huang. Evolving requirements-to-code trace links across versions of a software system. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 99–109, 2016. (Cited on page 165.)

B. Ramesh, L. Cao, and R. Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010. ISSN 1365-2575. (Cited on page 24.)

S. Reddivari, S. Rad, T. Bhowmik, N. Cain, and N. Niu. Visual Requirements Analytics: A Framework and Case Study. *Requirements Engineering*, 19(3): 257–279, 2014. (Cited on pages 140 and 156.)

M. Rees. A Feasible User Story Tool for Agile Software Development? In *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC)*, pages 22–30, 2002. (Cited on pages 13, 76, and 136.)

T. Reinhard, S. Meier, and M. Glinz. An Improved Fisheye Zoom Algorithm for Visualizing and Editing Hierarchical Models. In *Proceedings of the International Workshop on Requirements Engineering Visualization (REV)*. IEEE, 2007. (Cited on page 156.)

P. Rempel, P. Mäder, and T. Kuschke. Towards feature-aware retrieval of refinement traces. In *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 100–104, 2013. (Cited on page 165.)

M. Robeer, G. Lucassen, J. M. E. M. v. d. Werf, F. Dalpiaz, and S. Brinkkemper. Automated extraction of conceptual models from user stories via nlp. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 196–205, 2016. (Cited on pages 21, 103, 107, 108, 118, 119, 121, 131, 135, 146, 158, 163, and 174.)

W. N. Robinson. Integrating Multiple Specifications Using Domain Goals. *SIGSOFT Software Engineering Notes*, 14(3):219–226, 1989. (Cited on page 55.)

K. Rouibah and S. Al-Rafee. Requirement engineering elicitation methods: A kuwaiti empirical study about familiarity, usage and perceived value. *Information Management & Computer Security*, 17(3):192–217, 2009. (Cited on page 40.)

E. Rubin and H. Rubin. Supporting Agile Software Development through Active Documentation. *Requirements Engineering*, 16(2):117–132, 2010. (Cited on page 108.)

P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2): 131–164, 2009. ISSN 1382-3256. (Cited on page 85.)

K. Ryan. The Role of Natural Language in Requirements Engineering. In *Proceedings of the International Symposium on Requirements Engineering (ISRE)*, pages 240–242. IEEE, 1993. (Cited on pages 7, 14, 45, 59, 65, 78, 106, and 137.)

R. Saavedra, L. Ballejos, and M. Ale. Software requirements quality evaluation: State of the art and research challenges. In *Proceedings of the Argentine Symposium on Software Engineering (ASSE)*, 2013. (Cited on pages 8 and 14.)

M. Saeki, H. Horai, and H. Enomoto. Software Development Process from Natural Language Specification. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 64–73. ACM, 1989. (Cited on pages 10, 138, and 157.)

V. B. R. V. Sagar and S. Abirami. Conceptual Modeling of Natural Language Functional Requirements. *Journal of Systems and Software*, 88:25–41, 2014. ISSN 0164-1212. (Cited on pages 9, 10, 15, 102, 110, 139, 141, and 158.)

P. Sawyer, I. Sommerville, and S. Viller. Requirements process improvement through the phased introduction of good practice. *Software Process: Improvement and Practice*, 3(1):19–34, 1997. ISSN 1099-1670. (Cited on page 5.)

A. Schröter, T. Zimmermann, R. Premraj, and A. Zeller. If your bug database could talk. In *Proceedings of the International Symposium on Empirical Software Engineering (ISESE)*, pages 18–20, 2006. (Cited on page 86.)

E. Shihab, Z. M. Jiang, W. M. Ibrahim, B. Adams, and A. E. Hassan. Understanding the impact of code and process metrics on post-release defects: A case study on the Eclipse project. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 4:1–4:10. ACM, 2010. ISBN 978-1-4503-0039-1. (Cited on page 86.)

B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proceedings of the Symposium on Visual Languages (VL)*, pages 336–343, 1996. (Cited on pages 11, 107, 146, and 192.)

H. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, MA, 3rd edition, 1996. (Cited on page 46.)

J. Slankas and L. Williams. Automated Extraction of Non-Functional Requirements in Available Documentation. In *Proceedings of the International Workshop on Natural Language Analysis in Software Engineering (NaturaLiSE)*, pages 9–16, 2013. (Cited on page 79.)

H. F. Soares, N. S. R. Alves, T. S. Mendes, M. Mendonça, and R. O. Spínola. Investigating the Link between User Stories and Documentation Debt on Software Projects. In *Proceedings of the International Conference on Information Technology - New Generations (ITNG)*, pages 385–390, 2015. (Cited on page 137.)

M. Soeken, R. Wille, and R. Drechsler. Assisted behavior driven development using natural language processing. In C. A. Furia and S. Nanz, editors, *Proceedings of the 50th International Conference on Objects, Models, Components, Patterns (TOOLS)*, pages 269–287, 2012. ISBN 978-3-642-30561-0. (Cited on page 168.)

I. Sommerville and J. Ransom. An empirical study of industrial requirements engineering process assessment and improvement. *ACM Transactions on Software Engineering and Methodolology*, 14(1):85–117, 2005. ISSN 1049-331X. (Cited on pages 97, 98, and 100.)

Standish Group. The standish group 2016 chaos report. 2016. (Cited on page 5.)

J. Tague-Sutcliffe. The Pragmatics of Information Retrieval Experimentation, Revisited. *Information Processing & Management*, 28(4):467—490, 1992. (Cited on pages 69 and 89.)

G. Taylor and C. Tingguag. Linguistic, cultural, and subcultural issues in contrastive discoure analysis: Anglo-american and chinese scientific texts. *Applied Linguistics*, 12(3):319, 1991. (Cited on page 195.)

T. J. Teorey, G. Wei, D. L. Bolton, and J. A. Koenig. ER Model Clustering as an Aid for User Communication and Documentation in Database Design. *Communications of the ACM*, 32(8):975–987, 1989. (Cited on page 157.)

A. M. Tjoa and L. Berger. Transformation of Requirement Specifications Expressed in Natural Language into an EER Model. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, volume 823 of *LNCS*, pages 206–217. Springer, 1993. (Cited on pages 110, 111, and 113.)

S. F. Tjong and D. M. Berry. The Design of SREE: A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, volume 7830 of *LNCS*, pages 80–95. Springer, 2013. (Cited on pages 58 and 78.)

A. Trask, P. Michalak, and J. Liu. sense2vec-A Fast and Accurate Method for Word Sense Disambiguation in Neural Word Embeddings. *arXiv preprint arXiv:1511.06388*, 2015. (Cited on pages 11, 146, and 192.)

M. Trkman, J. Mendling, and M. Krisper. Using Business Process Models to better Understand the Dependencies among User Stories. *Information and Software Technology*, 71:58 – 76, 2016. ISSN 0950-5849. (Cited on page 136.)

A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. ISSN 00264423, 14602113. URL http://www.jstor.org/stable/2251299. (Cited on page 6.)

Y. Tzitzikas and J.-L. Hainaut. How to Tame a Very Large ER Diagram (Using Link Analysis and Force-directed Drawing Algorithms). In *Proceedings of the International Conference on Conceptual Modeling (ER)*, pages 144–159. 2005. (Cited on page 157.)

M. Unterkalmsteiner, T. Gorschek, A. K. M. M. Islam, C. K. Cheng, R. B. Permadi, and R. Feldt. Evaluation and measurement of software process improvement-a systematic literature review. *IEEE Transactions on Software Engineering*, 38(2):398–424, 2012. ISSN 0098-5589. (Cited on page 97.)

spacy.io. spaCy: Build Tomorrow's Language Technologies, 2016. URL http://spacy.io. (Cited on page 144.)

S. Vajjala and D. Meurers. Readability-based sentence ranking for evaluating text simplification. *ArXiv e-prints: 1603.06009*, 2016. (Cited on page 89.)

T. Vale, E. S. de Almeida, V. Alves, U. Kulesza, N. Niu, and R. de Lima. Software product lines traceability: A systematic mapping study. *Information and Software Technology*, 2016. ISSN 0950-5849. (Cited on page 162.)

S. Van Lingen, A. Palomba, and G. Lucassen. On the software ecosystem health of open source content management systems. In *Proceedings of the International Workshop on Software Ecosystems (IWSECO)*, pages 38–49, 2013.

C. Vehlow, F. Beck, and D. Weiskopf. The State of the Art in Visualizing Group Structures in Graphs. In R. Borgo, F. Ganovelli, and I. Viola, editors, *Eurographics Conference on Visualization (EuroVis) - STARs*. The Eurographics Association, 2015. (Cited on page 159.)

M. Vela and T. Declerck. A Methodology for Ontology Learning: Deriving Ontology Schema Components from Unstructured Text. In *Proceedings of the Workshop on Semantic Authoring, Annotation and Knowledge Markup (SAAKM)*, pages 22–26, 2009. (Cited on pages 110 and 113.)

Version One. 11th annual state of agile survey. Technical report, Technical report, Version One, 2017. (Cited on page 2.)

B. Wake. INVEST in Good Stories, and SMART Tasks. http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/, 2003. Accessed: 2015-02-18. (Cited on pages 9, 14, 29, 44, 48, 54, and 57.)

J. Wang and Q. Wang. Analyzing and Predicting Software Integration Bugs Using Network Analysis on Requirements Dependency Network. *Requirements Engineering*, 21(2), 2016. (Cited on page 108.)

X. Wang, L. Zhao, Y. Wang, and J. Sun. The Role of Requirements Engineering Practices in Agile Development: An Empirical Study. In *Proceedings of APRES*, volume 432 of *LNCS*, pages 195–209. 2014. (Cited on pages 3, 24, 39, 44, 75, 82, 108, 136, 146, and 164.)

J. H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. (Cited on page 145.)

Y. Wautelet, S. Heng, M. Kolp, and I. Mirbel. Unifying and Extending User Story Models. In *Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE)*, volume 8484 of *LNCS*, pages 211–225. 2014. (Cited on pages 2, 14, 39, 45, 47, 77, 104, 114, and 144.)

Y. Wautelet, S. Heng, D. Hintea, M. Kolp, and S. Poelmans. Bridging User Story Sets with the Use Case Model. In S. Link and J. C. Trujillo, editors, *Proceedings of ER Workshops*, pages 127–138. 2016a. (Cited on pages 137 and 139.)

Y. Wautelet, S. Heng, M. Kolp, and C. Scharff. Towards an Agent-driven Software Architecture Aligned with User Stories. In *Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART)*, pages 337–345, 2016b. (Cited on pages 137 and 139.)

J. Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, Jan. 1966. ISSN 0001-0782. (Cited on page 6.)

J. Westland. The cost of errors in software development: evidence from industry. *Journal of Systems and Software*, 62(1):1 – 9, 2002. ISSN 0164-1212. (Cited on page 5.)

N. Wilde and M. C. Scully. Software reconnaissance: Mapping program features to code. *Journal of Software Maintenance: Research and Practice*, 7(1):49–62, 1995. ISSN 1096-908X. (Cited on page 172.)

C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer, 2012. (Cited on page 18.)

M. Wynne and A. Hellesoy. *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Bookshelf, 2012. (Cited on pages 166 and 168.)

X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie. Automated Extraction of Security Policies from Natural-language Software Documents. In *Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, pages 12:1–12:11. ACM, 2012. (Cited on page 79.)

H. Yang, A. De Roeck, V. Gervasi, A. Willis, and B. Nuseibeh. Speculative Requirements: Automatic Detection of Uncertainty in Natural Language Requirements. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 11–20, 2012. (Cited on page 78.)

R. K. Yin. *Case Study Research - Design and Methods*. SAGE Publications, 2009. (Cited on page 17.)

C. Yu and H. Jagadish. Schema Summarization. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 319–330, 2006. (Cited on page 157.)

E. S.-K. Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, 1996. (Cited on page 137.)

E. S. K. Yu and J. Mylopoulos. Understanding "Why" in Software Process Modelling, Analysis, and Design. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 159–168. IEEE, 1994. (Cited on pages 4 and 42.)

X. Yuan, X. Li, M. Yu, X. Cai, Y. Zhang, and Y. Wen. Summarizing Relational Database Schema Based on Label Propagation. In *Web Technologies and Applications*, pages 258–269. 2014. (Cited on page 157.)

T. Yue, L. C. Briand, and Y. Labiche. A Systematic Review of Transformation Approaches between User Requirements and Analysis Models. *Requirements Engineering*, 16(2):75–99, 2010. (Cited on pages 11 and 138.)

T. Yue, L. C. Briand, and Y. Labiche. aToucan: An Automated Framework to Derive UML Analysis Models from Use Case Models. *ACM Transactions on Software Engineering and Methodology*, 24(3):13:1–13:52, 2015. ISSN 1049-331X. (Cited on pages 138 and 158.)

P. Zave and M. Jackson. Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, 1997. (Cited on pages 33, 53, and 72.)

Y. Zhang, C. Wan, and B. Jin. An empirical study on recovering requirement-to-code links. In *Proceedings of the International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 121–126, 2016. (Cited on page 164.)

T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for Eclipse. In *Proceedings of the International Workshop on Predictor Models in Software Engineering (PROMISE)*, 2007. (Cited on page 86.)

D. Zowghi and V. Gervasi. On the Interplay between Consistency, Completeness, and Correctness in Requirements Evolution. *Information and Software Technology*, 45(14):993 – 1009, 2003. ISSN 0950-5849. (Cited on page 77.)

All Published Work by Garm Lucassen

1. G. Lucassen, M. Robeer, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Extracting conceptual models from user stories with visual narrator. *Requirements Engineering*, 22(3):339–358, 2017c

2. G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Improving user story practice with the grimm method: A multiple case study in the software industry. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 235–252, 2017b

3. G. Lucassen, F. Dalpiaz, J. M. E. M. v. d. Werf, S. Brinkkemper, and D. Zowghi. Behavior-driven requirements traceability via automated acceptance tests. In *Proceedings of the International Workshop on Just-In-Time Requirements Engineering (JIT-RE)*, pages 431–434, 2017a

4. G. Lucassen, M. van de Keuken, F. Dalpiaz, S. Brinkkemper, G. W. Sloof, and J. Schlingmann. Jobs-to-be-done oriented requirements engineering: A method for defining job stories. In Submission

5. N. Bik, G. Lucassen, and S. Brinkkemper. A reference method for user story requirements in agile systems development. In *Proceedings of the International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 292–298, 2017

6. G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Visualizing user story requirements at multiple granularity levels via semantic relatedness. In *Proceedings of the International Conference on Conceptual Modeling (ER)*, pages 463–478, 2016c

7. G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Improving Agile Requirements: The Quality User Story Framework and Tool. *Requirements Engineering*, 21(3):383–403, 2016b

8. G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. The Use and Effectiveness of User Stories in Practice. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 205–222, 2016a

9. M. Robeer, G. Lucassen, J. M. E. M. v. d. Werf, F. Dalpiaz, and S. Brinkkemper. Automated extraction of conceptual models from user stories via nlp. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 196–205, 2016

10. P. Lombriser, F. Dalpiaz, G. Lucassen, and S. Brinkkemper. Gamified requirements engineering: Model and experimentation. In *Proceedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)*, pages 171–187, 2016

11. B. Arendse and G. Lucassen. Toward tool mashups: Comparing and combining nlp re tools. In *Proceedings of the International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, pages 26–31, 2016

12. G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper. Forging High-Quality User Stories: Towards a Discipline for Agile Requirements. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 126–135, 2015

13. G. Lucassen, J. M. E. M. van der Werf, and S. Brinkkemper. Alignment of software product management and software architecture with discussion models. In *Proceedings of the International Workshop on Software Product Management (IWSPM)*, pages 21–30, 2014

14. G. Lucassen and S. Jansen. Gamification in consumer marketing - future or fallacy? *Procedia - Social and Behavioral Sciences*, 148:194 – 202, 2014. International Conference on Strategic Innovative Marketing (ICSIM)

15. G. Lucassen, K. van Rooij, and S. Jansen. Ecosystem health of cloud paas providers. In *Proceedings of the International Conference on Software Business (ICSOB)*, pages 183–194, 2013

16. S. Van Lingen, A. Palomba, and G. Lucassen. On the software ecosystem health of open source content management systems. In *Proceedings of the International Workshop on Software Ecosystems (IWSECO)*, pages 38–49, 2013

17. G. Lucassen, S. Brinkkemper, S. Jansen, and E. Handoyo. Comparison of visual business modeling techniques for software companies. In *Proceedings of the International Conference on Software Business (ICSOB)*, pages 79–93, 2012

Summary

Contemporary movies like The Social Network would lead you to believe that multi-billion software companies such as Facebook are built on individual genius. In reality, complex software is created by teams of software professionals that each have their own personality profile and expertise: from highly technical software engineers to business-minded salespeople or artistic user experience experts. The challenge? The entire team needs to talk about and agree on what piece of the software puzzle to create next.

To facilitate and capture discussion on new software to be built, 50% of software companies have adopted a lightweight requirements approach called *user stories*: short descriptions of something a piece of software is supposed to do, told from the perspective of the person who desires the new capability. Most user stories follow a strict, compact template that captures *who* it is for, *what* it expects from the system, and (optionally) *why* it is important. Although many different templates exist, 70% of practitioners use the Connextra template: *"As a ⟨role⟩ , I want ⟨goal⟩, [so that ⟨benefit⟩]"*.

Despite this recent and substantial transition by industry, academic studies on user stories were essentially non-existent at the start of this research. Indeed, little is known about how user stories are actually used in requirements engineering practice, and it is unclear whether contemporary requirements engineering research results can successfully be applied to user stories. Strange, as their concise and simple textual structure makes user stories ideal for taking advantage of computational linguistics to try to solve many requirements engineering problems. With this in mind, this dissertation investigates the topic of user stories from the inside out. By reading this dissertation, you will learn more about why user stories are popular, how to support creating high-quality user stories, and, most importantly, how to help practitioners in fully achieving the title of this dissertation: *"Understanding User Stories"*.

The first three chapters of this dissertation seek to explain why user stories are popular and how to help practitioners in creating high-quality user stories. Prompted by the discovery that 56% of user stories made by practitioners include preventable errors and that guidelines for user stories quality significantly increase practitioners' productivity and work deliverable quality, this dissertation proposes the Quality User Story framework and accompanying natural language processing tool Automatic Quality User Story Artisan. By

taking advantage of the concise and well-structured nature of high-quality user stories, AQUSA detects a subset of QUS' quality defects with 92% recall and 77% precision.

The remaining three chapters focus on helping practitioners increase their understanding of their user stories. We demonstrate that it is possible to take advantage of computational linguistics techniques to extract and visualize the most important concepts from a collection of user stories. Our resulting tools, the Visual Narrator and Interactive Narrator, provide requirements engineers with a holistic overview of the system without the need for any human effort, supporting their detection of requirements ambiguity and conflicts. By taking advantage of user stories' focus on the essential components of a requirement *who? what? why?*, we achieve state-of-the-art accuracy between 88% recall and 91% precision up to 98% recall and 97% precision. Furthermore, we propose a holistic approach to agile requirements engineering with user stories: the Grimm User Story Method. Figure 8.3 presents a simplified overview of how Grimm generates specialized visualizations from a user story collection by connecting our three RE NLP tool contributions: AQUSA, Visual Narrator and Interactive Narrator.

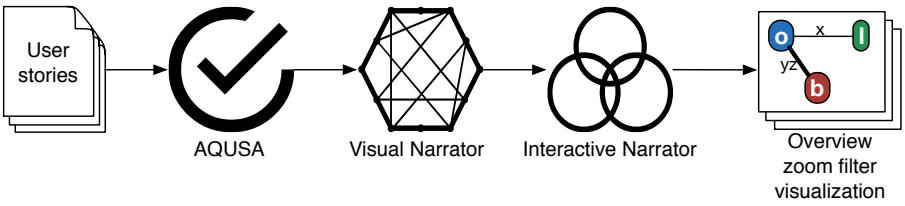


Figure 8.3: Simplified version of the Grimm User Story Method

Samenvatting

Films als The Social Network doen geloven dat een gigantisch softwarebedrijf als Facebook alleen kan ontstaan wanneer een geniale einzelgänger zichzelf maandenlang opsluit met zijn computer. In de èchte wereld ontstaat complexe software pas wanneer een team van software professionals effectief samenwerkt. Het is echter nog een flinke uitdaging om technische ontwikkelaars, zakelijke verkopers en kunstzinnige user experience experts op één lijn te krijgen.

Om teamdiscussie te stimuleren én te documenteren maakt bijna de helft van alle softwarebedrijven gebruik van een lichtgewicht requirements engineering techniek genaamd *user stories*: korte beschrijvingen van wat een stuk software zou moeten doen, verteld vanuit het perspectief van de persoon die deze nieuwe capaciteit graag wil. De meeste user stories volgen een strict, compact sjabloon om over een capaciteit vast te leggen voor *wie* het is, *wat* het verwacht van het systeem, en, optioneel, waarom het belangrijk is. Hoewel er veel verschillende sjablonen zijn, gebruikt 70% van software professionals de Connextra template: *"As a ⟨role⟩ , I want ⟨goal⟩, [so that ⟨benefit⟩]"*.

Ondanks deze substantiële populariteit, was er in 2014 nog nauwelijks onderzoek gedaan naar deze relatief nieuwe techniek. We weten nog maar weinig over hoe user stories nou gebruikt worden in de praktijk en het is onduidelijk of recent requirements engineering onderzoek ook van toepassing is op user stories. Opmerkelijk, aangezien de beknopte en simpele tekstuele structuur van user stories ze bijzonder geschikt maakt om diverse requirements engineering problemen op te lossen met computationele linguïstische methoden. Na het lezen van dit proefschrift weet je meer over waarom user stories populair zijn, hoe je user stories van hoge kwaliteit formuleert en hoe je de titel van dit proefschrift behaalt: *"User Stories Doorgronden"*.

In de eerste drie hoofdstukken van dit proefschrift verklaren we de populariteit van user stories en onderzoeken we hoe hoge kwaliteit user stories te formuleren. De ontdekking dat kwaliteitsrichtlijnen voor user stories de productiviteit en werkkwaliteit van software professionals verbeteren, terwijl 56% van user stories uit het bedrijfsleven nog fouten bevatten, is voor ons de aanleiding om het Quality User Story framework en de bijbehorende tool Automatic Quality User Story Artisan te presenteren om veelgemaakte fouten eenvoudig te kunnen voorkomen. Door gebruik te maken van de voorspelbare structuur van gedegen user stories, detecteert AQUSA een gedeelte van QUS

kwaliteitsdefecten met 92% recall en 77% precision.

In de laatste drie hoofdstukken onderzoeken we hoe we software professionals kunnen helpen hun user stories beter te doorgronden. We demonstreren dat het mogelijk is om gebruik te maken van computationele linguïstische methoden om de belangrijkste informatie uit user stories te destilleren en te visualiseren. Requirements engineers gebruiken de door ons ontwikkelde tools Visual Narrator en Interactive Narrator om automatisch een functioneel overzicht van het systeem te genereren zodat zij gemakkelijk ambiguïteit en conflicten in requirements kunnen vinden. Door gebruik te maken van user stories' focus op de essentie van een requirement *wie? wat? waarom?*, behalen we *state-of-the-art* resultaten tussen 88%-98% recall en 91%-97% precision. Daarnaast presenteren we onze holistische aanpak voor agile requirements engineering met User Stories: de Grimm User Story Method. Hieronder staat een versimpeld overzicht van hoe Grimm onze drie RE NLP tools combineert om gespecialiseerde visualisaties te genereren (Figuur 8.4).

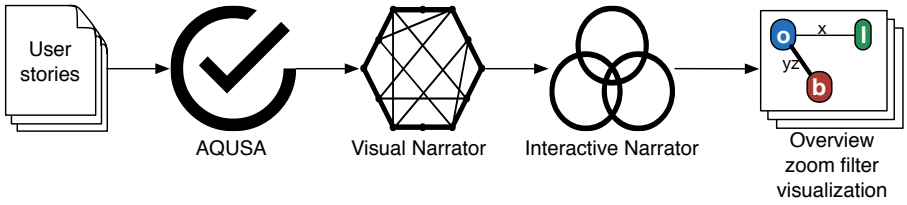


Figure 8.4: Versimpeld overzicht van de Grimm User Story Method

Curriculum Vitae

Garm Lucassen was born on June 10th, 1991, in Voorburg, the Netherlands. Between 2009 and 2014, he attended Utrecht University, studying to obtain a Bachelor's degree in Information Science and a Master's degree in Business Informatics. His Master's thesis was on the interplay between Software Product Management and Software Architecture, which sparked his interest in the central role of requirements for both of these disciplines.

In 2014, Garm Lucassen continued his research as a PhD candidate in software product management at the Center for Organization and Information, Department of Information and Computing Sciences, Utrecht Unversity. Soon after, his discovery that very little academic literature is available on *user stories* launched his investigation into this lightweight requirements engineering technique. During his PhD, Garm Lucassen coordinated and taught courses on software product management to professionals twice a year, and to students Business Informatics once a year. Furthermore, he is a board member of the International Software Product Management Association and contributes to the organization of industry and scientific conferences such as the Software Product Summit, the International Working Conference on Requirements Engineering: Foundation for Software Quality and the International Workshop on Software Product Management.

SIKS Dissertation Series

2011

01 Botond Cseke (RUN), Variational Algorithms for Bayesian Inference in Latent Gaussian Models

02 Nick Tinnemeier (UU), Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

03 Jan Martijn van der Werf (TUE), Compositional Design and Verification of Component-Based Information Systems

04 Hado van Hasselt (UU), Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference

05 Bas van der Raadt (VU), Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

06 Yiwen Wang (TUE), Semantically-Enhanced Recommendations in Cultural Heritage

07 Yujia Cao (UT), Multimodal Information Presentation for High Load Human Computer Interaction

08 Nieske Vergunst (UU), BDI-based Generation of Robust Task-Oriented Dialogues

09 Tim de Jong (OU), Contextualised Mobile Media for Learning

10 Bart Bogaert (UvT), Cloud Content Contention

11 Dhaval Vyas (UT), Designing for Awareness: An Experience-focused HCI Perspective

12 Carmen Bratosin (TUE), Grid Architecture for Distributed Process Mining

13 Xiaoyu Mao (UvT), Airport under Control. Multiagent Scheduling for Airport Ground Handling

14 Milan Lovric (EUR), Behavioral Finance and Agent-Based Artificial Markets

15 Marijn Koolen (UvA), The Meaning of Structure: the Value of Link Evidence for Information Retrieval

16 Maarten Schadd (UM), Selective Search in Games of Different Complexity

17 Jiyin He (UVA), Exploring Topic Structure: Coherence, Diversity and Relatedness

18 Mark Ponsen (UM), Strategic Decision-Making in complex games

19 Ellen Rusman (OU), The Mind's Eye on Personal Profiles

20 Qing Gu (VU), Guiding service-oriented software engineering - A view-based approach

21 Linda Terlouw (TUD), Modularization and Specification of Service-Oriented Systems

22 Junte Zhang (UVA), System Evaluation of Archival Description and Access

23 Wouter Weerkamp (UVA), Finding People and their Utterances in Social Media

24 Herwin van Welbergen (UT), Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

25 Syed Waqar ul Qounain Jaffry (VU), Analysis and Validation of Models for Trust Dynamics

26 Matthijs Aart Pontier (VU), Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

27 Aniel Bhulai (VU), Dynamic website optimization through autonomous management of design patterns

28 Rianne Kaptein (UVA), Effective Focused Retrieval by Exploiting Query Context and Document Structure

29 Faisal Kamiran (TUE), Discrimination-aware Classification

30 Egon van den Broek (UT), Affective Signal Processing (ASP): Unraveling the mystery of emotions

31 Ludo Waltman (EUR), Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

32 Nees-Jan van Eck (EUR), Methodological Advances in Bibliometric Mapping of Science

33 Tom van der Weide (UU), Arguing to Motivate Decisions

34 Paolo Turrini (UU), Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

35 Maaike Harbers (UU), Explaining Agent Behavior in Virtual Training

36 Erik van der Spek (UU), Experiments in serious game design: a cognitive approach

37 Adriana Burlutiu (RUN), Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

38 Nyree Lemmens (UM), Bee-inspired Distributed Optimization

39 Joost Westra (UU), Organizing Adaptation using Agents in Serious Games

40 Viktor Clerc (VU), Architectural Knowledge Management in Global Software Development

41 Luan Ibraimi (UT), Cryptographically Enforced Distributed Data Access Control

42 Michal Sindlar (UU), Explaining Behavior through Mental State Attribution

43 Henk van der Schuur (UU), Process Improvement through Software Operation Knowledge

44 Boris Reuderink (UT), Robust Brain-Computer Interfaces

45 Herman Stehouwer (UvT), Statistical Language Models for Alternative Sequence Selection

46 Beibei Hu (TUD), Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

47 Azizi Bin Ab Aziz (VU), Exploring Computational Models for Intelligent Support of Persons with Depression

48 Mark Ter Maat (UT), Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

49 Andreea Niculescu (UT), Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012

01 Terry Kakeeto (UvT), Relationship Marketing for SMEs in Uganda

02 Muhammad Umair (VU), Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

03 Adam Vanya (VU), Supporting Architecture Evolution by Mining Software Repositories

04 Jurriaan Souer (UU), Development of Content Management System-based Web Applications

05 Marijn Plomp (UU), Maturing Interorganisational Information Systems

06 Wolfgang Reinhardt (OU), Awareness Support for Knowledge Workers in Research Networks

07 Rianne van Lambalgen (VU), When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

08 Gerben de Vries (UVA), Kernel Methods for Vessel Trajectories

09 Ricardo Neisse (UT), Trust and Privacy Management Support for Context-Aware Service Platforms

10 David Smits (TUE), Towards a Generic Distributed Adaptive Hypermedia Environment

11 J.C.B. Rantham Prabhakara (TUE), Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

12 Kees van der Sluijs (TUE), Model Driven Design and Data Integration in Semantic Web Information Systems

13 Suleman Shahid (UvT), Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

14 Evgeny Knutov (TUE), Generic Adaptation Framework for Unifying Adaptive Web-based Systems

15 Natalie van der Wal (VU), Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

16 Fiemke Both (VU), Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

17 Amal Elgammal (UvT), Towards a Comprehensive Framework for Business Process Compliance

18 Eltjo Poort (VU), Improving Solution Architecting Practices

19 Helen Schonenberg (TUE), What's Next? Operational Support for Business Process Execution

20 Ali Bahramisharif (RUN), Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

21 Roberto Cornacchia (TUD), Querying Sparse Matrices for Information Retrieval

22 Thijs Vis (UvT), Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

23 Christian Muehl (UT), Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

24 Laurens van der Werff (UT), Evaluation of Noisy Transcripts for Spoken Document Retrieval

25 Silja Eckartz (UT), Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

26 Emile de Maat (UVA), Making Sense of Legal Text

27 Hayrettin Gurkok (UT), Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

28 Nancy Pascall (UvT), Engendering Technology Empowering Women

29 Almer Tigelaar (UT), Peer-to-Peer Information Retrieval

30 Alina Pommeranz (TUD), Designing Human-Centered Systems for Reflective Decision Making

31 Emily Bagarukayo (RUN), A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

32 Wietske Visser (TUD), Qualitative multi-criteria preference representation and reasoning

33 Rory Sie (OUN), Coalitions in Cooperation Networks (COCOON)

34 Pavol Jancura (RUN), Evolutionary analysis in PPI networks and applications

35 Evert Haasdijk (VU), Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

36 Denis Ssebugwawo (RUN), Analysis and Evaluation of Collaborative Modeling Processes

37 Agnes Nakakawa (RUN), A Collaboration Process for Enterprise Architecture Creation

38 Selmar Smit (VU), Parameter Tuning and Scientific Testing in Evolutionary Algorithms

39 Hassan Fatemi (UT), Risk-aware design of value and coordination networks

40 Agus Gunawan (UvT), Information Access for SMEs in Indonesia

41 Sebastian Kelle (OU), Game Design Patterns for Learning

42 Dominique Verpoorten (OU), Reflection Amplifiers in self-regulated Learning

43 Withdrawn

44 Anna Tordai (VU), On Combining Alignment Techniques

45 Benedikt Kratz (UvT), A Model and Language for Business-aware Transactions

46 Simon Carter (UVA), Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

47 Manos Tsagkias (UVA), Mining Social Media: Tracking Content and Predicting Behavior

48 Jorn Bakker (TUE), Handling Abrupt Changes in Evolving Time-series Data

49 Michael Kaisers (UM), Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

50 Steven van Kervel (TUD), Ontologogy driven Enterprise Information Systems Engineering

51 Jeroen de Jong (TUD), Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

2013

01 Viorel Milea (EUR), News Analytics for Financial Decision Support

02 Erietta Liarou (CWI), MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

03 Szymon Klarman (VU), Reasoning with Contexts in Description Logics

04 Chetan Yadati (TUD), Coordinating autonomous planning and scheduling

05 Dulce Pumareja (UT), Groupware Requirements Evolutions Patterns

06 Romulo Goncalves (CWI), The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

07 Giel van Lankveld (UvT), Quantifying Individual Player Differences

08 Robbert-Jan Merk (VU), Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

09 Fabio Gori (RUN), Metagenomic Data Analysis: Computational Methods and Applications

10 Jeewanie Jayasinghe Arachchige (UvT), A Unified Modeling Framework for Service Design.

11 Evangelos Pournaras (TUD), Multi-level Reconfigurable Self-organization in Overlay Services

12 Marian Razavian (VU), Knowledge-driven Migration to Services

13 Mohammad Safiri (UT), Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly

14 Jafar Tanha (UVA), Ensemble Approaches to Semi-Supervised Learning Learning

15 Daniel Hennes (UM), Multiagent Learning - Dynamic Games and Applications

16 Eric Kok (UU), Exploring the practical benefits of argumentation in multi-agent deliberation

17 Koen Kok (VU), The PowerMatcher: Smart Coordination for the Smart Electricity Grid

18 Jeroen Janssens (UvT), Outlier Selection and One-Class Classification

19 Renze Steenhuizen (TUD), Coordinated Multi-Agent Planning and Scheduling

20 Katja Hofmann (UvA), Fast and Reliable Online Learning to Rank for Information Retrieval

21 Sander Wubben (UvT), Text-to-text generation by monolingual machine translation

22 Tom Claassen (RUN), Causal Discovery and Logic

23 Patricio de Alencar Silva (UvT), Value Activity Monitoring

24 Haitham Bou Ammar (UM), Automated Transfer in Reinforcement Learning

25 Agnieszka Anna Latoszek-Berendsen (UM), Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

26 Alireza Zarghami (UT), Architectural Support for Dynamic Homecare Service Provisioning

27 Mohammad Huq (UT), Inference-based Framework Managing Data Provenance

28 Frans van der Sluis (UT), When Complexity becomes Interesting: An Inquiry into the Information eXperience

29 Iwan de Kok (UT), Listening Heads

30 Joyce Nakatumba (TUE), Resource-Aware Business Process Management: Analysis and Support

31 Dinh Khoa Nguyen (UvT), Blueprint Model and Language for Engineering Cloud Applications

32 Kamakshi Rajagopal (OUN), Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development

33 Qi Gao (TUD), User Modeling and Personalization in the Microblogging Sphere

34 Kien Tjin-Kam-Jet (UT), Distributed Deep Web Search

35 Abdallah El Ali (UvA), Minimal Mobile Human Computer Interaction

36 Than Lam Hoang (TUe), Pattern Mining in Data Streams

37 Dirk Börner (OUN), Ambient Learning Displays

38 Eelco den Heijer (VU), Autonomous Evolutionary Art

39 Joop de Jong (TUD), A Method for Enterprise Ontology based Design of Enterprise Information Systems

40 Pim Nijssen (UM), Monte-Carlo Tree Search for Multi-Player Games

41 Jochem Liem (UVA), Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning

42 Léon Planken (TUD), Algorithms for Simple Temporal Reasoning

43 Marc Bron (UVA), Exploration and Contextualization through Interaction and Concepts

---

2014

01 Nicola Barile (UU), Studies in Learning Monotone Models from Data

02 Fiona Tuliyano (RUN), Combining System Dynamics with a Domain Modeling Method

03 Sergio Raul Duarte Torres (UT), Information Retrieval for Children: Search Behavior and Solutions

04 Hanna Jochmann-Mannak (UT), Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation

05 Jurriaan van Reijsen (UU), Knowledge Perspectives on Advancing Dynamic Capability

06 Damian Tamburri (VU), Supporting Networked Software Development

07 Arya Adriansyah (TUE), Aligning Observed and Modeled Behavior

08 Samur Araujo (TUD), Data Integration over Distributed and Heterogeneous Data Endpoints

09 Philip Jackson (UvT), Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language

10 Ivan Salvador Razo Zapata (VU), Service Value Networks

11 Janneke van der Zwaan (TUD), An Empathic Virtual Buddy for Social Support

12 Willem van Willigen (VU), Look Ma, No Hands: Aspects of Autonomous Vehicle Control

13 Arlette van Wissen (VU), Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains

14 Yangyang Shi (TUD), Language Models With Meta-information

15 Natalya Mogles (VU), Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare

16 Krystyna Milian (VU), Supporting trial recruitment and design by automatically interpreting eligibility criteria

17 Kathrin Dentler (VU), Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability

18 Mattijs Ghijsen (UVA), Methods and Models for the Design and Study of Dynamic Agent Organizations

19 Vinicius Ramos (TUE), Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support

20 Mena Habib (UT), Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

21 Kassidy Clark (TUD), Negotiation and Monitoring in Open Environments

22 Marieke Peeters (UU), Personalized Educational Games - Developing agent-supported scenario-based training

23 Eleftherios Sidirourgos (UvA/CWI), Space Efficient Indexes for the Big Data Era

24 Davide Ceolin (VU), Trusting Semi-structured Web Data

25 Martijn Lappenschaar (RUN), New network models for the analysis of disease interaction

26 Tim Baarslag (TUD), What to Bid and When to Stop

27 Rui Jorge Almeida (EUR), Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty

28 Anna Chmielowiec (VU), Decentralized k-Clique Matching

29 Jaap Kabbedijk (UU), Variability in Multi-Tenant Enterprise Software

30 Peter de Cock (UvT), Anticipating Criminal Behaviour

31 Leo van Moergestel (UU), Agent Technology in Agile Multiparallel Manufacturing and Product Support

32 Naser Ayat (UvA), On Entity Resolution in Probabilistic Data

33 Tesfa Tegegne (RUN), Service Discovery in eHealth

34 Christina Manteli (VU), The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.

35 Joost van Ooijen (UU), Cognitive Agents in Virtual Worlds: A Middleware Design Approach

36 Joos Buijs (TUE), Flexible Evolutionary Algorithms for Mining Structured Process Models

37 Maral Dadvar (UT), Experts and Machines United Against Cyberbullying

38 Danny Plass-Oude Bos (UT), Making brain-computer interfaces better: improving usability through post-processing.

39 Jasmina Maric (UvT), Web Communities, Immigration, and Social Capital

40 Walter Omona (RUN), A Framework for Knowledge Management Using ICT in Higher Education

41 Frederic Hogenboom (EUR), Automated Detection of Financial Events in News Text
42 Carsten Eijckhof (CWI/TUD), Contextual Multidimensional Relevance Models
43 Kevin Vlaanderen (UU), Supporting Process Improvement using Method Increments
44 Paulien Meesters (UvT), Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.
45 Birgit Schmitz (OUN), Mobile Games for Learning: A Pattern-Based Approach
46 Ke Tao (TUD), Social Web Data Analytics: Relevance, Redundancy, Diversity
47 Shangsong Liang (UVA), Fusion and Diversification in Information Retrieval

---

2015

01 Niels Netten (UvA), Machine Learning for Relevance of Information in Crisis Response
02 Faiza Bukhsh (UvT), Smart auditing: Innovative Compliance Checking in Customs Controls
03 Twan van Laarhoven (RUN), Machine learning for network data
04 Howard Spoelstra (OUN), Collaborations in Open Learning Environments
05 Christoph Bösch (UT), Cryptographically Enforced Search Pattern Hiding
06 Farideh Heidari (TUD), Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
07 Maria-Hendrike Peetz (UvA), Time-Aware Online Reputation Analysis
08 Jie Jiang (TUD), Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
09 Randy Klaassen (UT), HCI Perspectives on Behavior Change Support Systems
10 Henry Hermans (OUN), OpenU: design of an integrated system to support lifelong learning
11 Yongming Luo (TUE), Designing algorithms for big graph datasets: A study of computing bisimulation and joins
12 Julie M. Birkholz (VU), Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
13 Giuseppe Procaccianti (VU), Energy-Efficient Software
14 Bart van Straalen (UT), A cognitive approach to modeling bad news conversations
15 Klaas Andries de Graaf (VU), Ontology-based Software Architecture Documentation
16 Changyun Wei (UT), Cognitive Coordination for Cooperative Multi-Robot Teamwork
17 André van Cleeff (UT), Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
18 Holger Pirk (CWI), Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
19 Bernardo Tabuenca (OUN), Ubiquitous Technology for Lifelong Learners
20 Lois Vanhée (UU), Using Culture and Values to Support Flexible Coordination
21 Sibren Fetter (OUN), Using Peer-Support to Expand and Stabilize Online Learning
22 Zhemin Zhu (UT), Co-occurrence Rate Networks
23 Luit Gazendam (VU), Cataloguer Support in Cultural Heritage

24 Richard Berendsen (UVA), Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation

25 Steven Woudenberg (UU), Bayesian Tools for Early Disease Detection

26 Alexander Hogenboom (EUR), Sentiment Analysis of Text Guided by Semantics and Structure

27 Sándor Héman (CWI), Updating compressed colomn stores

28 Janet Bagorogoza (TiU), Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO

29 Hendrik Baier (UM), Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains

30 Kiavash Bahreini (OU), Real-time Multimodal Emotion Recognition in E-Learning

31 Yakup Koç (TUD), On the robustness of Power Grids

32 Jerome Gard (UL), Corporate Venture Management in SMEs

33 Frederik Schadd (TUD), Ontology Mapping with Auxiliary Resources

34 Victor de Graaf (UT), Gesocial Recommender Systems

35 Jungxao Xu (TUD), Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

---

2016

01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines

02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow

03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support

04 Laurens Rietveld (VU), Publishing and Consuming Linked Data

05 Evgeny Sherkhonov (UVA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers

06 Michel Wilson (TUD), Robust scheduling in an uncertain environment

07 Jeroen de Man (VU), Measuring and modeling negative emotions for virtual training

08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data

09 Archana Nottamkandath (VU), Trusting Crowdsourced Information on Cultural Artefacts

10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms

11 Anne Schuth (UVA), Search Engines that Learn from Their Users

12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems

13 Nana Baah Gyan (VU), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach

14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization

15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments

16 Guangliang Li (UVA), Socially Intelligent Autonomous Agents that Learn from Human Reward

17 Berend Weel (VU), Towards Embodied Evolution of Robot Organisms

18 Albert Meroño Peñuela (VU), Refining Statistical Data on the Web

19 Julia Efremova (Tu/e), Mining Social Structures from Genealogical Data

20 Daan Odijk (UVA), Context & Semantics in News & Web Search

21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground

22 Grace Lewis (VU), Software Architecture Strategies for Cyber-Foraging Systems

23 Fei Cai (UVA), Query Auto Completion in Information Retrieval

24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach

25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior

26 Dilhan Thilakarathne (VU), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains

27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media

28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control

29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning

30 Ruud Mattheij (UvT), The Eyes Have It

31 Mohammad Khelghati (UT), Deep web content monitoring

32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations

33 Peter Bloem (UVA), Single Sample Statistics, exercises in learning from just one example

34 Dennis Schunselaar (TUE), Configurable Process Trees: Elicitation, Analysis, and Enactment

35 Zhaochun Ren (UVA), Monitoring Social Media: Summarization, Classification and Recommendation

36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies

37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry

38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design

39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect

40 Christian Detweiler (TUD), Accounting for Values in Design

41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance

42 Spyros Martzoukos (UVA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora

43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice

44 Thibault Sellam (UVA), Automatic Assistants for Database Exploration

45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control

46 Jorge Gallego Perez (UT), Robots to Make you Happy

47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks

48 Tanja Buttler (TUD), Collecting Lessons Learned

49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis

50 Yan Wang (UVT), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

---

2017

01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime

02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation

03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines

04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store

05 Mahdieh Shadi (UVA), Collaboration Behavior

06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search

07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly

08 Rob Konijn (VU), Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery

09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text

10 Robby van Delden (UT), (Steering) Interactive Play Behavior

11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment

12 Sander Leemans (TUE), Robust Process Mining with Guarantees

13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology

14 Shoshannah Tekofsky (UvT), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior

15 Peter Berck (RUN), Memory-Based Text Correction

16 Aleksandr Chuklin (UVA), Understanding and Modeling Users of Modern Search Engines

17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution

18 Ridho Reinanda (UVA), Entity Associations for Search

19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval

20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility

21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)

22 Sara Magliacane (VU), Logics for causal inference under uncertainty

23 David Graus (UVA), Entities of Interest — Discovery in Digital Traces

24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning

25 Veruska Zamborlini (VU), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search

26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch

27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors

28 John Klein (VU), Architecture Practices for Complex Contexts

29 Adel Alhuraibi (UvT), From IT-Business Strategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT

30 Wilma Latuny (UvT), The Power of Facial Expressions

31 Ben Ruijl (UL), Advances in computational methods for QFT calculations

32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives

33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity

34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics

35 Martine de Vos (VU), Interpreting natural science spreadsheets

36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging

37 Alejandro Montes Garcia (TUE), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy

38 Alex Kayal (TUD), Normative Social Applications

39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR

40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems

41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle

42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets

43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval

44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering

45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement

46 Jan Schneider (OU), Sensor-based Learning Support