Quality Metrics for Sustainability

The Operational Energy Use of Application Software

R. Spauwen

J.M. van der Werf

S. Brinkkemper

E. Jagroep

L. Blom

Technical Report UU-CS-2015-008
April 2015

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Utrecht University
Department of Information and Computer Science

Quality Metrics for Sustainability

The Operational Energy Use of Application Software

Technical Report

Ruvar Spauwen

Supervisors:
Jan Martijn van der Werf
Sjaak Brinkkemper
Erik Jagroep
Leen Blom

Utrecht, April 2015

Abstract

In recent years, Information Technology (IT) has grown into a sector that is both as vital and, or possibly even more, resource consuming as Aviation. Combined with our ever-growing dependence of IT, the resulting energy costs as well as the inevitable depletion of materials, such as fossil fuels and rare earth materials, have driven the need for more sustainable solutions, both environmentally and economically. Consequently, the body of knowledge on *IT Sustainability*, or *Green IT*, is steadily growing, which has mainly yielded solutions for making hardware less resource costly and possibly more energy efficient. However, software can be seen as the true consumer of energy.

As the design of software greatly determines the specific utilization of hardware components, during operation, software architects have the potential to influence the sustainability of software. However, this potential is yet inaccessible due to the lack of architectural tactics that address sustainability, let alone proper metrics to determine and validate the energy consumption of software. Hence, this research attempts to develop metrics for the *operational energy use* of application software and, subsequently, apply them to determine relations between software design decisions, system workloads and impacts on the energy use of software products, independently of systems.

To achieve our primary objective, we measured the power consumption of different computer systems during *base*, *application* and *maximum* workload intervals, while in the meantime we also recorded the performance of the systems. The performance data was used afterwards to determine whether a specific measurement was acceptably clean (e.g. minimal amount of noise). Then, after collecting a sufficient amount of clean measurements, we applied a subtractive method to filter out unrelated base power consumptions and determine the operational energy use of an application.

Two applications were used in this research to develop the metrics. First, we performed initial experiments with a CPU-intensive stress test application, which has a relatively simple workload and power usage profile. After this, we further evaluated the metrics by performing a case-study with a document batch processing application, which has a more complex design and deployment. To improve the reliability of the measurements, we remotely operated and monitored the test systems and we performed multiple iterations, or *runs*, per specific measurement to minimize the influence of noise. In addition, we considered different configurations, or *versions*, of the test applications to enable the analysis of architectural decisions, such as single vs. dual-core processing.

Based on our findings, we propose a set of metrics that can be used to determine more precisely which portion of the system's energy usage belongs to the execution of an application. Also, the metrics show that, although the total amount of energy which different systems require to execute an application may differ greatly, the operational energy use of the application can still be fairly similar. Lastly, it shows that multi-core processing often requires more power yet less total energy.

Contents

1 Introduction **1**
1.1 Problem Statement 1
1.2 Research Objectives 2
1.3 Research Questions 3
1.4 Research Context 4
1.5 Thesis Content 6

2 Research Approach **7**
2.1 Research Design 7
2.2 Research Model 8
2.3 Research Method 9
2.4 Research Deliverables 13
2.5 Research Questions linked to PDD Deliverables 14

3 Towards Sustainable Software **15**
3.1 Sustainability and IT 15
3.2 Defining Green IT 16
3.3 Realizing Green IT Principles 17
3.4 Focusing on Green Software 17
3.5 The Role of Software Architecture 18

4 Current Practices **20**
4.1 Literature on Software Sustainability 20
4.2 Architects on Software Sustainability 24
4.3 Synthesizing the Current Practices 26

5 Building Metrics for Sustainability **29**
5.1 Recap of the Research Context 29
5.2 Measuring the Operational Energy Use 30

6 Experimentation **35**
6.1 Experimental Setup 35
6.2 Experimentation Results 40
6.3 Experimentation Analysis 43
6.4 Additional considerations 51

7 Validation **54**
7.1 Validation Setup 54
7.2 Validation Results 56
7.3 Validation Analysis 58

8 Discussion **66**
8.1 Findings & Research Contribution . . . 66
8.2 Research Limitations . . . 67

9 Conclusion **70**
9.1 General Conclusion . . . 70
9.2 Thesis Results Linked to RQs . . . 70
9.3 Future Research . . . 71

Bibliography **76**

List of Figures **78**

List of Tables **79**

Appendix A Data Collector Sets **80**
A.1 Experimentation Phase . . . 80
A.2 Validation Phase . . . 82

Appendix B Scripts **83**
B.1 PiBatch - Management System . . . 83
B.2 PiBatch - Application System . . . 85
B.3 Pressure - CPU Energy Use Profile . . . 86

Appendix C Structured Literature Review **87**
C.1 Search Term String . . . 87
C.2 Remaining Articles . . . 87

Appendix D Figures & Tables **88**
D.1 Workloads at Processes and CPUs during S^A and S^B . . . 88
D.2 Process-Deliverable Diagram . . . 90

Chapter 1

Introduction

Sustainability has become an important topic in the ever-growing field of information technology (IT), or the "digital economy", due to its consumption of huge amounts of resources, which consists of at least a tenth of the world's electricity use according to [38]. The recent emergence of *Green IT* has yielded many fruitful solutions in support of Sustainable Development (SD) in general and, more specifically, in addressing the vast amounts of resources (e.g., energy and materials) consumed by IT. In majority, the primary focus has been on achieving IT energy consumption savings by means of more energy efficient hardware [31]. However, software can be seen as the true consumer of energy, as it is software that often determines the use of the hardware, and thus the required energy [55]. Consequently, to control the energy consumption of the hardware, the software needs to become sustainable as well. Sustainable software is "software whose direct and indirect negative impacts on economy, society, human being and the environment resulting from development, deployment and usage of the software is minimal and/or has a positive effect on sustainable development [42]. In other words, sustainability has become an important non-functional requirement for software, and needs to be taken into account during the design of the software.

A software design depends highly on the mapping of requirements into its architecture, i.e. the quality of the software is determined by the degree to which the architecture satisfies the requirements [24]. According to [3], the term software architecture (SA) is defined as the discipline of creating the "structures of the software, which comprises software elements, the externally visible properties of those elements, and the relationships among them". An important class of externally visible properties are the non-functional requirements, which are often called quality attributes. A quality attribute is "a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders" [3]. As sustainability becomes an important aspect of the software architecture, it becomes a quality attribute of the software. Based on the quality attributes software needs to satisfy, the architect makes design decisions, called tactics. The effect of tactics (e.g. [47]) should be measurable. Thus, to test whether sustainability tactics have an effect, one needs a set of metrics that allows for comparison between different architectures.

1.1 Problem Statement

As recent literature reviews on sustainability in software engineering and green IT show, hardly any sustainability metrics touch the architectural aspects of software [5, 45]. As a consequence, architects cannot beforehand assess whether their design meets the desired sustainability (business) goals and requirements. Moreover, many of the results found tend to focus on the development aspect of sustainability [10], and tend to neglect the deployment and usage of the software, as in many software projects the main cost come from the development and maintenance [42]. Furthermore, there is yet no clear evidence as to whether the usage of software has a tangible impact on the amount of energy consumed by the executing system, i.e. the *operational energy use* [10, 50].

We can distinct between different types of software: application software, system software (e.g. the operating system, drivers, GUIs), and development software. These software types have different purposes and work on different layers. Although, for instance, the operating system is also an topic of interest with regards to sustainability, this thesis focuses on the application software. Application software is a set of one or more programs designed to carry out operations for a specific application. This type of software cannot run on itself and thus is dependent on the system software. Within application software, we distinct between tailor-made software and product software. Product software is "a packaged configuration of software components, or a software-based service with auxiliary materials, which is released for and traded in a specific market" [66], i.e., it is deployed at many different sites. Therefore, improving the operational energy use of product software has a high impact on its sustainability, not only for the organization that is developing the software, but also for the client at which the software is deployed or hosted, and that uses it.

One way to come to an objective measure for sustainability is to measure the energy use of the software itself; reducing the amount of energy the software consumes directly improves the sustainability of the software. However, current metrics typically focus on the quality (cf. [4]) or the method for measuring energy usages [2]. For instance, [33] proposes a subtractive method for measuring the energy use of specific components in a single system. Moreover, [27] proposes a generic metric to measure the energy-efficiency of software as well as a method to apply it in a software engineering process. This metric, which seamlessly integrates into the GREENSOFT Model (cf. [42, 28]), is used to compare different applications and tasks by looking at the "useful work done", a principal introduced by [59]. Furthermore, it uses so-called *white box measurements* to be able to tell (more precisely) in which part of the software there is potential for energy savings. However, their approach does not (yet) properly consider that modern computer systems process multiple applications concurrently. Hence, it is not that straightforward to determine the actual energy use solely related to a specific application, let alone independent of different systems.

In summary, product (application) software runs at different clients, i.e. hardware with different energy consumption properties but also with different system workloads (e.g., due to client-specific active processes). Furthermore, modern computers can execute multiple (application) processes concurrently (and/or sequentially within the minimal granularity of the available measuring instrumentation). Therefore, a metric is required that is both **independent of the hardware used** and **additional system workload** to be able to cope with different environments as well as limited instrumentations, but still allows for comparison on the level of the software architecture.

Based on the arguments above, the formal problem statement of this thesis project is as follows:

Software **metrics** play a key role in the measuring and, thus, realization of quality attributes. However, there are no metrics for the **operational energy use** of application software.

1.2 Research Objectives

Driven by the goal of realizing a sustainability quality attribute, this research attempts to develop metrics for the operational energy use of (product) application software, validating whether the energy required for executing an application can be determined independent of the executing system.

For this, the following objectives need to be met: construct practical and reliable methods for (1) measuring the energy consumption of systems as well as the monitoring of their workloads, (2) isolating the energy consumption solely related to the execution of the application from the total energy consumption of the system, and (3) develop metrics, or quantitative measures, which provide comparable insight into the energy consumption of applications between different systems.

1.3 Research Questions

Based on the problem statement and objectives, the main research question is defined as follows:

RQ. **How can we determine the operational energy use of application software?**

Several steps need to be performed before we can answer the main research question. Therefore, the following sub-questions (SQs) have been formulated in support of performing these steps:

SQ. 0 *What are the current practices, both theoretical and practical, with regards to software architectural aspects for measuring and developing sustainable software?*

In order to construct a solid knowledge base for this thesis, first a literature study as well as a study into current practices at the research company is conducted. Subsequently, the results of both studies will be combined and analyzed to determine a set of candidate architectural aspects, metrics or guidelines, which can be implemented within the experimentation phase.

SQ. 1 *How can we reliably measure the energy use of a computer system, execute an application and concurrently monitor its workload, with minimal interference?*

As stated below SQ. 0, our aim is to investigate the influence of one ore more architectural aspects on the sustainability of an application (e.g. energy efficiency), by means of an experiment. Yet, before we can do this, we need to be able to actually measure the energy use solely related to the execution of an application. However, to achieve this, we first need to be able to: (1) reliably measure the energy use of the executing system, (2) monitor the workload of the system so that we can determine how much of the workload is related to the application, and (3) perform these things without creating too much interference or noise during the measurements. Furthermore, we need to find suitable test application(s) and systems which cooperate with the preceding statements.

SQ. 2 *How can we isolate the energy use solely related to the execution of an application (e.g., perform a specific task) from the measured energy use of the underlying system(s)?*

We plan to apply the subtractive method proposed by [33] to determine the energy use solely related to the test application. This entails that we need to determine the total energy use of a system during the execution of the test application, determine the average power use when the system is idle, and assure that, during the execution of the application, the system is not occupied by other significantly heavy processes (expect for the test application). Since most systems have variable workload, which is difficult to control e.g. due to background process, we need to take adequate measures to ensure the quality of the measurements and minimize noise.

SQ. 3 *How can we use system specific properties to normalize the measured energy usages of executions of a single application on different systems and make them comparable?*

Given that there are many different types of computer systems and each system has a unique (and varying) workload, it can very be difficult (or even impossible) to determine the expected energy use of an application independent of the executing system. Nevertheless, we attempt to investigate to what extend we can normalize the energy use values over different systems e.g. on the basis of known system properties. If this proves to be feasible, we can work towards additional metrics that enable the prediction of energy use values based on known system properties.

SQ. 4 *How do the metrics behave in environments with other applications and/or systems?*

Another important property of the metrics would be their external validity, e.g. whether the metrics are applicable in a different environment than the experimentation's. To test this, we plan to perform additional measurements with a more complex application, which has been developed by the research partner and which is deployed in a (more representative) client-server environment.

SQ. 5 *To what extend can the resulting metrics be used to determine the influence of software architectural aspects on the sustainability of software?*

Although the main part of the thesis is focused on the actual method and metrics of measuring of the operational energy use, we still aim to investigate the influence of an architectural aspect on the sustainability of an application. Therefore, we design the experiments and validation in such a way that we can include a candidate aspect, which follows from the current practices studies.

The presented research questions will be matched with the corresponding thesis findings within the conclusion of this document (see Section 9.2), i.e. the final research implications of this thesis.

1.4 Research Context

This section describes the involved stakeholders, scope, and scientific relevance of this research.

1.4.1 Stakeholders

We can distinguish three main stakeholders that might have affected and/or can be affected by the outcome of this thesis project:

Academics Both the scientific research fields of software architecture as well as Green IT (and Software) have been receiving increasing attention. However, it appears that the development of “greener” software by means of architectural design decisions is still underexposed. Furthermore, the availability of proper metrics for measuring the operational energy use of (application) software is still scarce, let alone that it can be done system independently.

Software companies The results of this research will support software companies, together with their partners, in finding better solutions for their customers with regard to the sustainability of their software products. In addition, Centric (i.e. the research company) affects the outcome of this research by providing access to their current practices. Furthermore, domain experts (e.g. software architects working at Centric) have assisted us by means of participating in interviews and brainstorm sessions and providing support during the experiments.

Government & Society The higher purpose of this research is to support software companies in more easily and effectively implementing sustainability related business goals, by means of the software architecture. By doing so, these companies gain more possibilities for managing and lowering the energy consumption of both their IT products and services. Subsequently, these possibilities enable businesses to comply more easily with environmental standards and regulations which are, or most likely will be, issued by their governments and/or societies.

1.4.2 Scope

As discussed previously, the quality (or satisfaction) of a software product and its requirements depends heavily on the software architecture. Furthermore, the definition of “software architecture” implies that the term can be used to denote three main concepts: (1) the high-level structures of a software system, (2) the discipline of creating such high-level structures and (3) the documentation of these high-level structures. Subsequently, all of these concepts play important roles in the process of designing a software architecture. Then, according to [22], the process of designing a SA constitutes of four main activities, which are performed iteratively and at different stages of the initial software development life-cycle. First, the architectural analysis serves to (1) define the problems the architecture must solve and (2) to present them into a set of architecturally significant requirements (ASR). Second, the architectural synthesis results in candidate architectural solutions to the set of ASRs. Third, the architecture evaluation activity ensures that the architectural decisions made are the right ones by measuring the candidate architectural solutions against the ASRs. Finally, the fourth activity is the architecture evolution, which entails the process of maintaining and adapting an existing SA to meet its requirements and environmental changes.

Initially, the scope of this research included two of the previously mentioned activities, namely: architectural analysis and architecture evaluation. The first activity (e.g., "define the problems the architecture must solve and present them as a set of ASRs") related to our goal of investigating how green software related business requirements could be translated into architectural aspects. Then, the second activity (e.g., "being able to measure the candidate architectural solutions against the ASRs") relates to our goal of being able to measure and validate the resulting aspects, ultimately. However, since the focus of the thesis has shifted primarily towards the operational energy use of software, the architectural analysis activity has become less significant with respect to the scope.

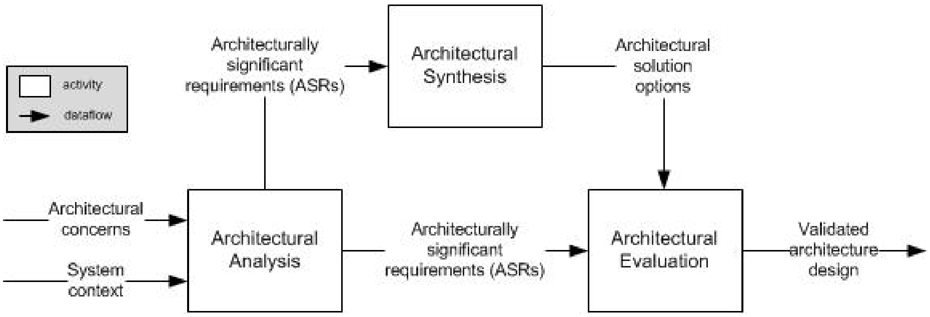

activity
dataflow
Architecturally significant requirements (ASRs)
Architectural Synthesis
Architectural solution options
Architectural concerns
System context
Architectural Analysis
Architecturally significant requirements (ASRs)
Architectural Evaluation
Validated architecture design

Figure 1.1: Core activities in Software Architecture Design, according to [22]

Continuing with the "sustainability" focus of this thesis, we mentioned before that Green IT includes both hardware, software and telecommunication. Moreover, Green IT can both be used to realize sustainability and IT itself can be made more sustainable (e.g. with regards to development, operations, maintenance). This thesis focuses on software and, more precisely, how itself can be made less resource consuming during it operation, i.e. the operational energy use of software.

Combining these two research fields, we eventually want to have architectural tactics which can be applied to realize sustainability requirements. However, we begin with metrics because we first need to be able to measure and evaluate the sustainability, before we can influence it effectively.

1.4.3 Scientific relevance

The scientific trigger for this research relates mainly to the emerging gap between the relatively mature research field of Software Architecture and the still evolving field of Green Software.

First, software architecture as a scientific research area has already been studied extensively by individual researchers as well as institutions. Much of this research is focused on defining and codifying the fundamental aspects of the discipline, such as architectural styles (e.g., patterns), architecture description languages, architecture documentation and formal methods [16]. The maturity of the research discipline has improved considerably in recent years with the introduction of several international standards, such as *ISO/IEC 42010* which has recently been revised to *ISO/IEC 42010:2011* [25]. This standard addresses the creation, analysis and sustainment of SAs through architecture descriptions (e.g. a foundation for expressing, communicating, and reviewing SAs). Also, it specifies the requirements that apply to architecture descriptions, frameworks and ADLs. In addition, there is also the *ISO quality standards for measuring architectures* developed by [32]. This research work in particular supports the current general insight that the architecture of a software system is closely related to its quality attributes.

Then, considering Green IT, it is clear that achieving SD through energy efficient hardware has much scientific relevance (e.g., the increasing number of research papers and conferences dedicated to the topic). However, scientific research dedicated to achieving SD through solely the software is still scarce and quantitative empirical research on a large scale is not yet available. Furthermore, those few research endeavors that do focus on the development of Green Software, such as [34, 30], have not sufficiently investigated the potential role of the SA. Then, [67] does consider the influence of the SA on the energy consumption of software e.g. by comparing different architecture styles for concurrency. However, this work mainly focuses on comparing different SA implementations and is therefore limited to a single system, without considering system independent metrics.

1.4.4 Social Relevance

The social trigger for this research relates to the increasing awareness regarding the “greenness” of software products, which is caused by rising energy costs, new governmental policies, requirements from customers, etc. The ultimate goal of this research is to contribute to solutions that support businesses in reacting to these issues more effectively and earlier. Its results (e.g. sustainability tactics and metrics) can support software companies with the realization of green business goals and requirements. Eventually, these actions can lead to decreasing the operational energy use of their products and, thus, lower energy costs. Other benefits can be: improving the quality of the software product, a better corporate image and a higher satisfaction from customers and society.

1.5 Thesis Content

Now that the research problem, objectives, questions and context have been introduced, this chapter concludes with an detailed overview of the remainder of this thesis document, i.e. the main content in which the presented research questions will be systemically addressed and answered.

In Chapter 2, we present the applied research approach of this thesis, which includes its research design, model and method, research deliverables, design principles, and the research evaluation.

Chapter 3 provides additional explanation on relevant theory, such as sustainability and ICT in general, the origins of Green IT and Green Software, and the role of the software architecture.

Chapter 4 comprises our studies of current practices. This includes the design and results of a structured literature review and brainstorm session, which was performed at the research company. Further, we discuss how we used the study results to come to a candidate architectural aspect and other considerations such as suitable test applications, systems, and reliable instrumentation.

In Chapter 5, we introduce the hypotheses that will be tested during the experimentation combined with overviews of the important terms and representations of the different measurements.

Chapter 6 contains the experimentation, which is divided into (1) the experimental setup, (2) results and (3) the analysis. Also, other experiment-related considerations are included at the end.

Then, in Chapter 7, the metrics resulting from Chapter 6 will be validated in an case-study, which basically entails conducting additional measurements with a more complex applications.

Chapter 8 discusses the findings of this thesis, combined with its contributions and limitations.

Finally, Chapter 9 concludes this thesis by providing a general overview of the final research results, how they match with the research questions, and our recommendation for future research.

The Appendices include, amongst others: the scripts for automating the experiment measurements, used data collector sets, and the papers which (partially) resulted from this thesis project.

Chapter 2

Research Approach

This chapter discusses the design, model and method that form our research approach. Further, it includes an overview of the main deliverables and how these are linked to the research questions.

2.1 Research Design

According to [21], most of the research in the Information System (IS) discipline has a relation to at least one of the following paradigms: *behavioral science* and *design science*. The first paradigm, behavioral science, addresses research through the development and justification of theories that explain or predict human and organizational phenomena all through the phases of the development and implementation of IT and IS. The latter paradigm, design science, seeks to extend the boundaries of human and organizational capabilities through the creation and evaluation of new and innovative IT artifacts [35]. In order to position and compare these paradigms, [21] introduced a conceptual framework, which has been republished in [20]. To position the research in this thesis, we constructed a custom version of this framework, which is depicted in Figure 2.1.

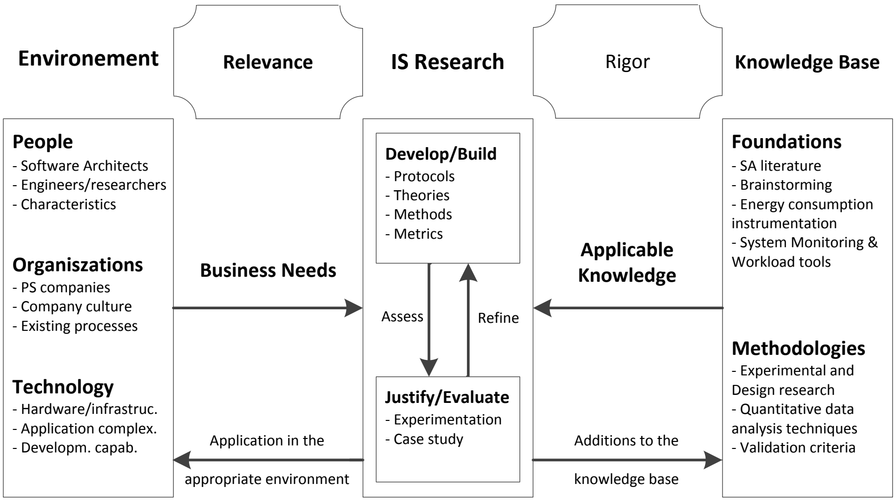


Figure 2.1: Research Design: the IS Research Conceptual Framework by [21]

The overall design of the framework entails that the relevance of the *IS research* is determined by business needs from the *Environment* and that applicable knowledge is required from the *Knowledge base* to assure its rigor. In return, the developed and evaluated research results should be applicable in the appropriate environment as well as provide additions to the knowledge base.

The *Environment* consists of the following concepts: people, (business) organizations and their existing or planned technology [52]. According to [53], the environment defines the problem space in which the phenomena of interest resides. Furthermore, it includes the goals, tasks, problems and opportunities that define business needs as they are perceived by people within the organization. The organizations in which this research is conducted are primarily product software companies and the people involved are software architects, engineers and researchers, which all have their own set of characteristics. In addition, factors such as the company culture (e.g. sustainability-mindedness) and existing processes influence the problem space. Finally, technology also plays an important role in the environment of this research. It includes the arrangement of hardware and infrastructure, but also the complexity of the existing applications and available development capabilities. The research in this thesis is set up in such a way that it meets the business needs that result from these concepts and, consequently, the relevance of the research can be assured.

The *Knowledge Base* "provides the raw materials from and through which IS research is accomplished" [21]. It is composed of the following two concepts: (1) foundations (e.g., theories, frameworks, models, methods), and (2) methodologies (e.g., data analysis techniques, formalisms, measures, validation criteria). The foundations can be used in the initial phase of the IS research, i.e. the *development* of theories within behavioral science or the *building* of artifacts within design science, respectively. Then, the methodologies can be used during the secondary phase, i.e. the *justification* of the theory or the *evaluation* of the artifact (e.g., case study, experimentation).

According to [21], the goal of behavioral science research is "truth", while the of goal of design science is "utility". Nevertheless, they argue that these goals are inseparable, since "truth informs design and utility informs theory". Hence, the research in this thesis is related to both paradigms. However, the main objective of this research is to produce a new and innovative artifact with the purpose of extending organizational and technological capabilities. For instance, a software metric will be build and evaluated. Therefore, this research mostly leans on the design science paradigm.

2.2 Research Model

The purpose of building a research model is to gain a more structured overview of the research approach. The model of this thesis, depicted in Figure 2.2, is based on the research model method designed by [63] as well as guidelines provided by [21]. Consequently, the model complies with the following rules: each component (i.e. rounded rectangle) represents a research object, the arrows indicate interaction between research objects, and the main research objects can be distinguished by their solid border. Furthermore, the model makes a distinction between the academic research objects, which receive the most attention in this thesis, and the business research objects, which are positioned at the far right side of the model. The academic research object are the following:

- A set of candidate metrics for sustainability is based on key findings from a **comprehensive study** of available scientific literature combined with an investigation of the current practice at Centric, the research company. The literature study includes the following topics: software architecture, requirements & metrics, Green IT, and, more specifically, Green Software.
- An **experimentation** is conducted to investigate the hypotheses that are build upon the study results. The initial experimentation includes multiple iterations of measurements, an application with a relatively simple design, and multiple application systems for testing.
- Based on the results of the initial experimentation, a **research paper** is written. This paper focuses on the metrics for the operational energy use of application software (See Appen. **??**).
- A **case-study** is conducted during which the final set of metrics are tested in a different environment (e.g. to test external validity). This includes performing additional measurements with a more complex application. Consequently, we can validate whether the metrics

work sufficiently with an application that has a more variable workload with respect to e.g. a specific system component and/or the distribution over all its system components.

In addition, three business research objects have been determined. First, an assessment of the current practices at the research company is provided (e.g., insight concerning to what extend, currently, software architects deal with sustainability related requirements). Second, although the metrics for the operational energy use of applications mostly lean towards the academic side of this thesis, they can already be valuable to certain companies that want to apply the metrics and guidelines to study their own products. Third, insight is provided into how software architectural decisions may influence the energy usage of application software (e.g. sustainability as a QA).

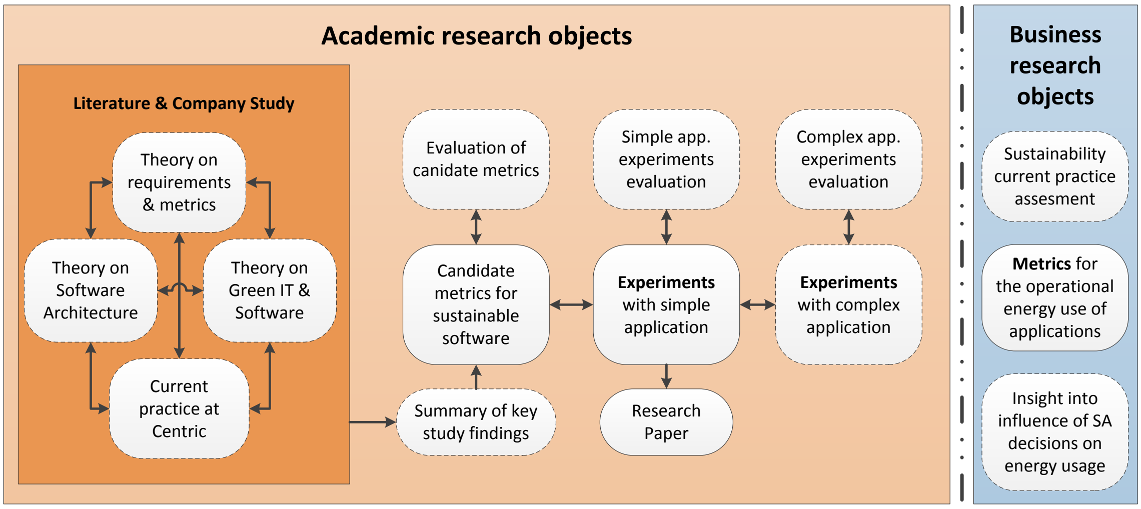

Academic research objects
Literature & Company Study
Theory on requirements & metrics
Theory on Software Architecture
Theory on Green IT & Software
Current practice at Centric
Evaluation of canidate metrics
Simple app. experiments evaluation
Complex app. experiments evaluation
Candidate metrics for sustainable software
Experiments with simple application
Experiments with complex application
Summary of key study findings
Research Paper
Business research objects
Sustainability current practice assesment
Metrics for the operational energy use of applications
Insight into influence of SA decisions on energy usage

Figure 2.2: Research Model

2.3 Research Method

Figures 2.3 depicts the detailed research method of this thesis, which is in the form of a process-deliverable diagram (PDD). The construction of a PDD is a meta-modeling technique that can be used for modeling activities and artifacts of a certain process. Further, it includes the activities that will be performed along with documents or research objects that will be delivered [62]. The main layout of the PDD consist of two integrated diagrams. The left part, i.e. the meta-process model, contains the various activities, or processes, related to the research as well as the sequence of their execution. The right part of the PDD, i.e. meta-data model, consists of the resulting research objects, or deliverables, and how they are related to each other. In addition, a PDD should be supported by an *Activity* table and a *Concept* table. The *Activity* table contains descriptions of the activities, sub-activities and their relations to the deliverables. The *Concept* table contains descriptions for the concepts introduced at the right side of the PDD, i.e. the meta-data model.

The PDD of this thesis has been constructed according to the conventions determined by [62]. The main *activities*, or *phases*, are visualized by dark gray-colored frames and the *sub-activities* are positioned within them. The main activities are to be executed in an sequential order by starting at the top of the diagram. Then, with respect to the sub-activities, the order of execution can be more varied. Certain sub-activities need to be performed in (semi-)concurrent order. This is visualized by two adjacent tracks of sub-activities. Furthermore, certain sub-activities need to be performed multiple number of times. This is visualized by an *if-then-else*-statement, in which the sub-activity will be repeated until the control-statement does not hold anymore. Finally, certain complex sub-activities (e.g. containing other sub-activities) are highlighted with a black shadow.

As Figure 2.3 shows, the PDD includes five main activities. Its design is similar to the general approach for design research developed by [58], which consists of the following five phases:

1. Awareness of the Problem
2. Suggestion
3. Development
4. Evaluation
5. Conclusion

The goal of the first phase, i.e. *Awareness of the Problem*, is to define the research trigger, or *problem*, together with the appropriate research approach. Since these elements have already been described in previous sections of this document, this phase has not been included in the PDD. Then, the purpose of the second phase, i.e. *Suggestion*, is to find possible answers to the proposed research questions. This phase correlates with the first activity in the PDD, i.e. the *Literature & Company Study.* During the second phase, i.e. *Development*, we will synthesize both studies and build our hypotheses (e.g. candidate metrics) upon them. These activities correlate partially with the *Development* phase, i.e. attempt to develop and implement the research artifacts according to the suggested solution, namely: the implementation of the candidate metrics. Consequently, the remainder of the *Experimentation* activity combined with the *Validation* activity correlate with the *Evaluation* phase, i.e. evaluate the resulting artifacts to confirm whether the solution solves the problem (e.g. with/without a new set of problems). Finally, the PDD's last main activity, i.e. *Conclusion*, correlates with the *Conclusion* phase. Each main activity is explained in more detail below. The related *Activity* and *Concept* tables have been included as appendices in Section D.2.

Literature & Company Study

This main activity contains two separate tracks, which are processed concurrently. The left track, i.e. the literature study, contains the sub-activities related to conducting a structured literature review. The right track, i.e. the company study, contains the sub-activities related to conducting a brainstorm session with domain experts. By conducting a literature review and interviews with domain experts, researchers are better able to identify key issues that are causing the specified problem. Furthermore, these activities enable the extraction of valuable knowledge (e.g., solutions from other areas, theories, ideas from stakeholders) which can be helpful for solving the problem.

Development

The *Development* activity consists of a single complex sub-activity, i.e. *Synthesize studies & build candidate metrics.* This sub-activity consists of multiple other sub-activities, which are explained in more detail in Section 4.3 and Chapter 5. The main deliverables of this activity are (1) to improve the knowledge base by synthesizing the preceding studies and (2) to build a set of hypotheses (e.g. the candidate metrics) that will be tested during the Experimentation activity.

Experimentation

This main activity contains the sub-activities for designing and conducting an experiment, which is an orderly procedure carried out with the goal of verifying, refuting, or establishing the validity of the hypotheses. For instance, when the candidate metrics have been constructed and the orientation prior to the experimentation is completed, the designed experiment should provide a controlled environment in which the metrics can be implemented, measured, validated and, if possible, improved. This activity contains multiple control-statements, which enables us to visualize the presence of loops in the model. Subsequently, these loops are used to visualize the presence of multiple measurement configurations during the experiment as well as checks with regards to whether a measurement was successfully executed. Furthermore, the main activity contains two complex activities, which consist of multiple sub-activities. Although other sub-activities also consist multiple other sub-activities, these two activities are modeled as "complex" to emphasize their magnitude (e.g. in required time and complexity) in comparison to the other activities.

During the first complex activity, i.e. *Develop experimental setup & protocol(s)*, we will use the experience gathered during the preceding orientation to come to a final experimental setup as well as the experiment protocols, which will be followed during the experiments. The experimental setup consist of, amongst others, the instrumentation (e.g. for measuring energy consumptions), the computers systems, and the test application. We use multiple systems to gather measurement data on different types of systems, which enables us to determine the quality of the metrics and to compare results. Furthermore, we use a separate system, i.e. the *Management System*, for remotely monitoring and operating the system on which the application is executed, i.e. the *Application System*. In addition, different configurations, or *versions*, of the test application will be included in the experiment to enrich our data collection and, eventually, to be able to compare different design decisions. Due to the exploratory nature of this research, we will re-evaluate both the experimental setup and protocols and improve them until both of them are satisfactory.

Measurement data will be gathered by performing the experiment protocol with different combinations of a system and an application version, i.e. an *AppSys*-configuration. Each time that we perform a measurement with a certain *AppSys*-configuration, we will evaluate the results and determine whether the measurement has completed successfully. If this is not the case, the measurement will be performed again, with improvements if necessary and possible. Otherwise, we will continue with the next uncompleted *AppSys*-configuration. When all the measurement data has been collected, the second complex activity follows: *Process measurement data & analyze results*. This is a complex activity, because a lot of measurement data will be produced and to process it all into analyzable results might take considerable amount of time. When the data processing and analysis is finished, a research paper will be written including the up until then achieved results.

Validation

In addition to the evaluations performed during the *Experimentation* activity, an additional evaluation is planned in the form of a case-study. This case-study takes place during the *Validation* activity, which consists of two complex sub-activities, i.e. *Perform measurements with DOCGEN* and *Process measurement data & analyze results*. The main differences between the measurements is that we use a single Application System and more complex test application, i.e. *DOCGEN*, during the latter main activity. The application will be provided by the research company. It has a client/server architecture and uses a remote database server. However, the main operations are processed on a single test server and therefore we focus on the energy use of this system. Then, the *Process measurement data & analyze results* activity is similar to the activity during the preceding phase: the measurements will produce a large amount of data which needs to be processed. In addition, we can evaluate to what extend the previously created methods and tooling for processing and analyzing the measurement data are applicable with the more complex application.

Conclusion

During the last main activity we will finalize the research results, which consist of the: the final metric(s), experimental setup and protocols, the experimentations and validation results, and the conclusions. Moreover, the final sub-activities include the reflection and discussion on the research results, identification of possibilities for future research and finalization of the thesis document.

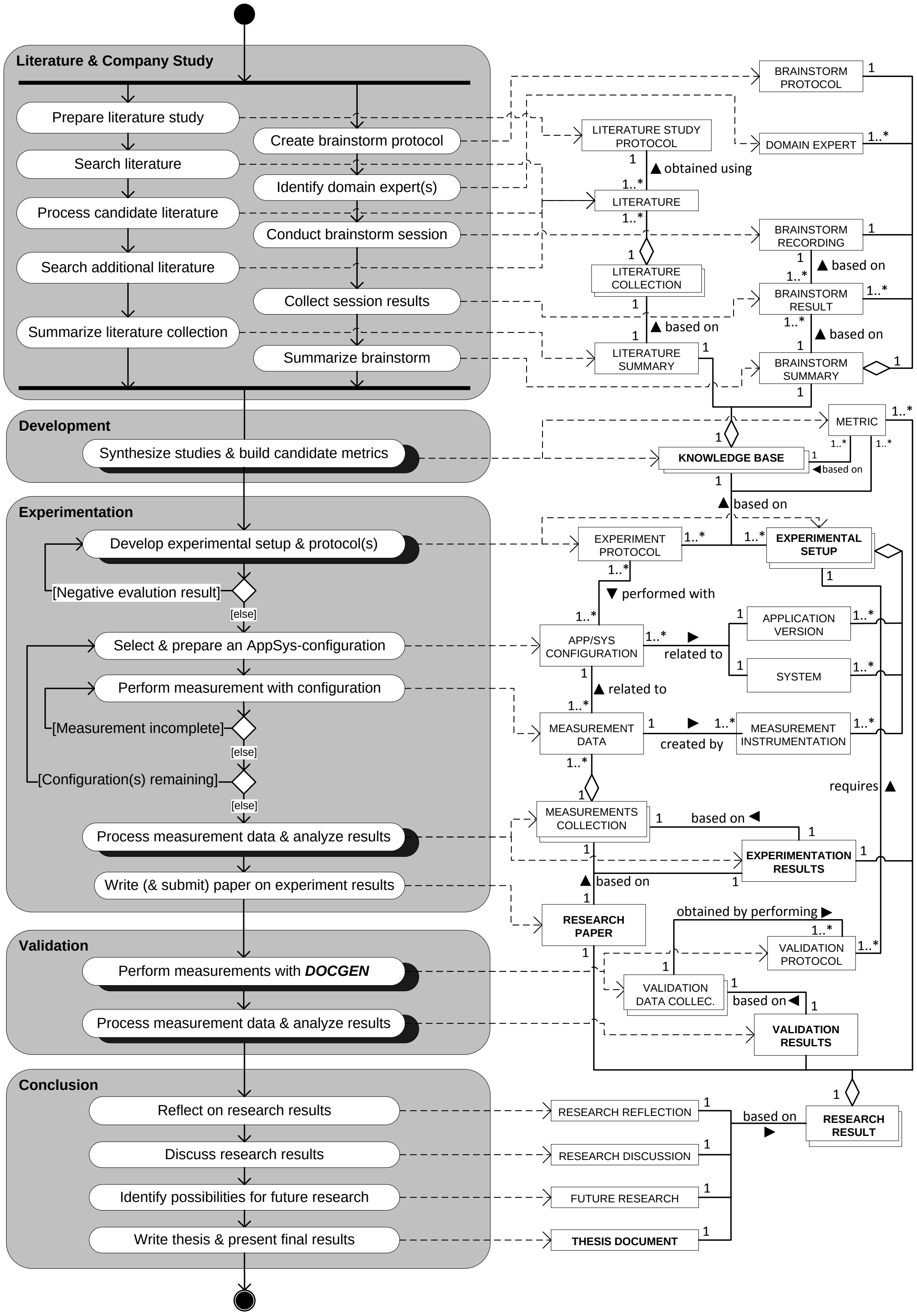


Figure 2.3: Process-Deliverable Diagram

2.4 Research Deliverables

Within this section, we present an overview and the descriptions of the main research deliverables:

Systematic literature review

A systematic literature review on the topic of software architectural metrics and aspects for sustainable software form the theoretical foundation of this research. However, as Section 4.1 will show, the applicability of the review results is limited due to the relative immaturity of the specific topic. Nevertheless, the results are used as guidance during both the company study (e.g. the method for developing a new quality attribute by [3]) and the construction of candidate metrics.

Set of candidate sustainability aspects

The brainstorm session with domain experts combined with the literature study results provide the necessary input to design a set of candidate sustainability (architectural) aspects. These aspects are analyzed to see which of them can be implemented in the experimentation phase. Only a subset of these aspects will be suitable (e.g. realistic and/or valuable) for implementation within in this thesis. Nevertheless, the complete set of candidate aspects, or a subset, can be a valuable foundation for future research.

Advice for experimental setups & experiment protocols

The preparation for and conducting of the experiments provide additional insights into the measuring of the energy use of both computer systems and application software. Furthermore, to support future research, the experimental setups and experiment protocols are described in great detail (including the many pitfalls someone most likely encounters when doing similar research).

Empirical data on the energy use and performance of systems and application software

Since there is only few research on the energy consumption of software (and even less is reliably measured or documented sufficiently), the empirical data created in this thesis will automatically be an important deliverable. Furthermore, through our extensive datasets, which is gathered with reliable instrumentation, we aim to provide sufficient insight into the strengths and weaknesses of our efforts (e.g. ensure their reliability), and to provide additional guidance for future research.

Metrics for the operational energy use of application software

The most important deliverable of this thesis will be the development and evaluation of a set of software metrics for the operational energy use of application software. In Section 5.2.2, we will introduce our hypotheses regarding this set of metrics. Then, during the Experimentation and Validation chapters, they will be investigated and evaluated in practice

Case-study & architectural aspect results

The case-study during the Validation will be of value to the research company, since it provides additional insight into the profile of their application. Furthermore, when future research is desired (with/without the case-study application), they have a proper foundation to build further on. In addition, the included architectural aspect results will provide an initial insight into the influence of certain SA design decisions on the energy consumption of the considered applications.

Research Paper & thesis document

The textual deliverables include the main documents resulting from this thesis, i.e. a research paper and thesis document. The paper will be submitted to a conference so that (if accepted) the research conducted during this thesis (e.g. ideas, methods and results) will be shared with others.

2.5 Research Questions linked to PDD Deliverables

Table 2.1 links the research questions introduced in Section 1.3 to the corresponding deliverables within the research method. This table shows how each deliverable will contribute to the research.

Table 2.1: Research Questions linked to Activities

Research Question	**Description**	**Contributing deliverables**
SQ0: *What are the current practices, both theoretical and practical, with regards to software architectural aspects (e.g. development and measuring) of sustainable software?*	The specific current practices are determined by synthesizing main findings of a literature study and a brainstorm session with domain experts.	• LITERATURE SUMMARY • BRAINSTORM SUMMARY • KNOWLEDGE BASE
SQ1: *How can we reliably measure the energy use of a computer system, execute an application and concurrently monitor its workload, with minimal interference?*	This question will be addressed by means of the experimental setup and the experiment protocols, which are build upon the knowledge base.	• KNOWLEDGE BASE • EXPERIMENTAL SETUP • EXPERIMENT PROTOCOL
SQ2: *How can we isolate the energy use solely related to the execution of an application (e.g., perform a specific task) from the measured energy use of the underlying system(s)?*	By applying the right instrumentation, we aim to collect sufficient and reliable data, which enables more detailed analysis.	• MEASUREMENT INSTRUMENTATION • MEASUREMENT DATA
SQ3: *How can we use system specific properties to normalize the measured energy usages of executions of a single application on different systems and make them comparable?*	We aim to use the instrumentation and data to determine which system specific properties can be used to overcome difference between the systems	• MEASUREMENT INSTRUMENTATION • MEASUREMENT DATA COLLECTION
SQ4: *How do the metrics behave in environments with other applications and/or systems?*	To further determine the external validity of the proposed metrics, we conduct additional measurements within a secondary experimental setup in which e.g. a more complex application is applied.	• EXPERIMENTAL SETUP • MEASUREMENT INSTRUMENTATION • VALIDATION DATA COLLECTION
SQ5: *To what extend can the resulting metrics be used to determine the influence of software architectural aspects on the sustainability of software?*	During the experimentation as well as the validation, we aim to include investigating the influence of an architectural aspects	• EXPERIMENTATION RESULTS • VALIDATION RESULTS

Chapter 3

Towards Sustainable Software

This chapter elaborates on the concepts of sustainability and IT, Green IT, and, especially, Green Software. Furthermore, it explains the concept of Software Architecture and why this concept, or process, is expected to play a key role in the realization of more sustainable software. When the reader is already familiar with one of these concepts, the corresponding section(s) can be skipped.

3.1 Sustainability and IT

Environmental sustainability has become an important topic in the ever-growing field of information (and communications) technology. A reason for this is that IT already consumes vast amounts of resources, such as energy, raw materials, and capital. For instance, it is estimated that the "digital economy" (i.e., the economy that is based on digital technologies) already accounts for at least a tenth of the world's electric energy consumption and more than 2 percent of global carbon emissions [38, 39]. Moreover, many begin to understand the importance of making IT more sustainable due to a limited amount (and thus a depleting availability) of natural resources, such as fossil fuels but also, for instance, computer component materials. This threat may lead to rising IT costs while in the meantime, we have become undeniably dependent of IT. Outside (but also partially inside) the field of IT, it is believed that sustainability can be achieved by balancing economic, environmental and social factors in equal harmony. Figure 3.1 illustrates this relationship by means of a Venn diagram in which the social, economic, and environmental factors overlap so as to produce a system that is sustainable, i.e. that it is socially bearable, economically equitable, and environmentally viable. Hence, in theory, true sustainability can be achieved by balancing all three factors. However, it is more often the case that the social and/or environmental factors are underexposed in comparison to the economic factor and, thus, no true sustainability is achieved [1].

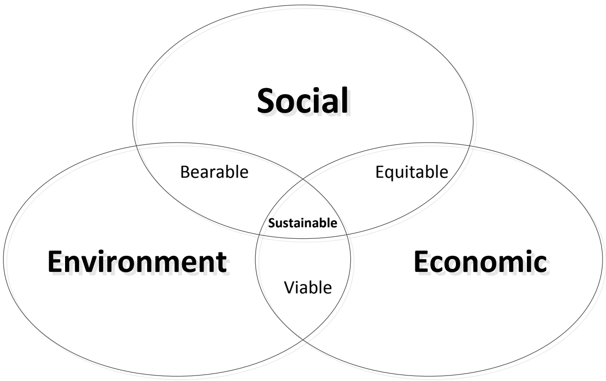


Figure 3.1: Venn diagram of Sustainability factors

In essence, Sustainability can be seen as an abstract destination while Sustainable Development (SD) is the pathway leading towards it. Although it still remains a vague concept for many, the most often-quoted definition for SD was already defined in 1987 by the Brudtland Commission: “development that meets the needs of the present without compromising the ability of future generations to meet their own needs” [8]. This implies that we need to look after our planet, our resources and our people to ensure that we can live in a sustainable manner and that we can hand down our planet to our children, and their children, to live in true sustainability. Besides sustainable development, businesses often use a Corporate Social Responsibility (CSR) business model to address their sense of responsibility towards the community and environment [40]. The term CSR became popular in the 1960s and one of its purposes is to exhort businesses to adopt non-financial measures of success, such as the Triple Bottom Line, i.e. an accounting framework which consists of 3 (already familiar) parts: social, environmental and economic [12]. These parts are also known as the “3 P’s” (i.e., people, planet and profit), or the “three pillars of Sustainability”.

3.2 Defining Green IT

Sustainability (as well as SD) with respect to IT consists of multiple aspects. On the one hand, we have the field of IT itself as the consumer of resources. But, on the other hand, IT can also be seen as an enabler of sustainability in many different industries, such as education, health care, and agriculture. These industries have both economical and environmental benefits from the use of IT and, in fact, many of them have become dependent of it. In other words, the continuous growth of the IT sector has impacted the environment negatively (in terms of resources consumption and waste), but IT also enables and supports many activities in general as well as environmental sustainability related ones. Due to this significance, sustainability in the field of IT has become an important (and popular) concept and, consequently, multiple terms have emerged over the years, including among others *Green IT*, *Green ICT*, *Green Computing*, and *I(C)T Sustainability.* The most common of these terms, Green IT, has been defined as “the study and practice of designing, manufacturing, using and disposing of computer hardware, software and communications systems efficiently and effectively with no or minimal impact on the environment” [41].

During the last decade, there has been a growing number of organizations that have adapted to the principles of Green IT as they are being triggered by the potential of lowering energy costs, decreasing carbon footprints, and improving corporate images. In addition, organizations are also being encouraged (and sometimes forced) to adapt to Green IT principles by means of governmental standards and regulations with respect to Sustainability, such as subsidization, carbon footprint taxes, and increasing waste disposal costs. Back in 2007, it was estimated by research firm Gartner that by 2009 “more than one-third of IT organizations” would have “one or more environmental criteria in their top-6 buying criteria for IT-related goods and services”, and that by 2010 “50 percent of IT organizations” would have declared “an environmental imperative” [39]. These statements were re-affirmed in 2011, when Gartner estimated that “improving sustainability would become a top-5 priority for 60 percent of major Western European and North American CEOs by 2015” [46]. Moreover, in 2013, Gartner reported that most of the efforts are primarily focused on energy and resource efficiency, compliance, and carbon reporting, with an additional focus on overall sustainable performance measurement and sustainable product management.

Although the numbers presented above are just estimations, they underline that environmental sustainability has become a major objective for an increasing number of organizations. Furthermore, it appears that (1) Green IT has begun to play an important role for achieving this objective and (2) it is applicable for organization both inside and outside the IT industry. Nevertheless, in the following subsection we limit our focus to IT organizations, such as computer hardware manufactures and software companies, and explain how Green IT can be realized, among others.

3.3 Realizing Green IT Principles

According to [41], a holistic approach needs to be adopted in order to address the environmental impacts of IT, comprehensively and effectively. Moreover, [41] states that such and approach should be along the following 4 complementary paths: *green design*, *green manufacturing*, *green use*, and *green disposal*. The combined goal of these paths, or *fronts*, is to achieve total environmental sustainability from the IT side and make IT greener throughout its entire life cycle. Due to the broadness of IT, many different approaches have emerged already for, or "in support of", the realization of Green IT principles. Examples of such approaches are, amongst others: terminal servers combined with low power thin-clients, (better) recycling of computing equipment, and the (fruitful) process of making large data centers more energy efficient. Especially the latter gained a lot of attention due to the continuing growth of services provided through the WWW together with the increasing demand for more and better data centers. This attention resulted in several broad and effective measures for "greening" data centers, such as innovative Eco-friendly designs, alternative methods for conserving energy (e.g., recycling heat), and virtualization of servers [41].

Further, most approaches concerning the realization of Green IT principles prefer to follow the *green use* path, and, moreover, most of them have been directed on hardware-related aspects [7]. In addition, [41] states that "a key green objective in using computer systems and operating data centers is to reduce their energy consumption, thereby minimizing the greenhouse gas emissions" and we "can significantly reduce energy consumption by making small changes to the ways we use computers". For instance, we can enable (intelligent) power management features, more often turn off a system when it is not in use, and/or use more energy-efficient systems. However, the impact and practicality of these approaches can be limited and, moreover, sustainable features (e.g., built into hardware) might be eliminated by inefficiently designed and/or programmed software [54].

3.4 Focusing on Green Software

The previous section argues that, in contrast to the increased interest for the use of "greener" hardware, there are still hardly any (practical) approaches that aim to achieve SD solely through the software (e.g., by incorporating it in its design). According to [42], this is partially caused by a lack of models, descriptions, and realizations in the specific area. However, some positive changes are noticeable as recently different research works have been devoted to methods for measuring the energy consumption as well as the efficiency of software (e.g. [51, 67]) and models for making the software engineering process more sustainable (e.g. [34, 42]). For instance, Figure 3.2 shows a recent version of the GREENSOFT Model, which originates from [42]. This model can be used as a conceptual reference model, which helps to organize and classify research results, actions, framework, process models, etc. Furthermore, it includes a "cradle-to-grave product life cycle model for software products, sustainability metrics and criteria for software, software engineering extensions for sustainably sound software design and development, as well as appropriate guidance" [42].

In addition, [42] also introduced a definition for the concept of "Green Software", i.e. "software of which the direct and indirect negative impacts on economy, society, human being, and the environment resulting from development, deployment, and usage of the software is minimal". Following this definition, [57] states that Green Software is required to fulfill the following three abstract "green" factors: (1) how resource efficient is it to develop, maintain, and discontinue the software, (2) how resource efficient is the software during its execution, and (3) to what extend does it support sustainable development in general. Examples of resources required during the development and execution of software are, amongst others, raw material, human resources, or energy consumption. The latter resource is an important factor during the usage phase of software, since current systems often need to run over long periods of time, process large amounts of data, and/or have many users that execute the software. Subsequently, these aspects lead to more intensive and longer use of computer hardware. This, in turn, increases the direct operational costs

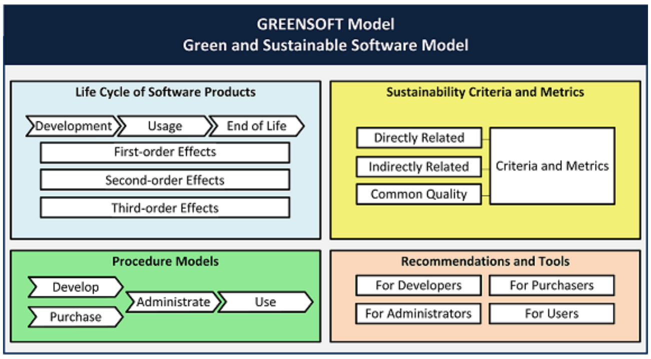


Figure 3.2: The GREENSOFT model by [42]

of the system and, also, more energy is required for cooling the hardware since more heat is being produced. Therefore, we can argue that in particular cases even small improvements with respect to the energy efficiency of software during its execution can lead to a significant decrease of the energy consumption of its related system. However, according to [29], making existing software more energy efficient or designing new green and sustainable software can be complex tasks.

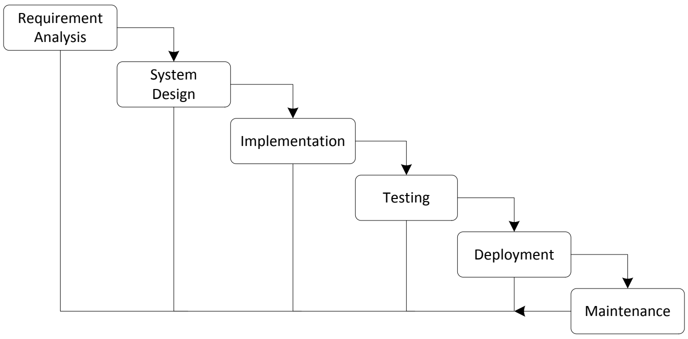


Figure 3.3: Waterfall Model of the SDLC based on [48]

As Figure 3.3 illustrates, a Software Development Life Cycle (SDLC) consist of multiple phases, i.e. requirements gathering and analysis, system design, implementation, testing, deployment, and maintenance. Consequently, when a novel business goal is introduced (e.g. sustainability), it needs to be taken into account during (most of) the SLDC in order to realize it properly. This includes, amongst others, the determination of additional requirements, how these requirements might influence other (existing) requirements (e.g. trade-offs), how the requirements can be implemented in the software design, and how the realization of these requirements eventually can be validated. The next section will continue by explaining which types of software requirements exist and why we should focus on the system design phase for the realization of sustainability related requirements.

3.5 The Role of Software Architecture

In general, requirements for software systems can be divided into two different types: functional requirements, which define what a system is supposed to do, and non-functional requirements (NFR), which specify criteria that judge how well a system performs its intended functions. In addition, the NFR type can also be divided into (at least) two sub-types: constraints (e.g., design decisions with zero degrees of freedom) and quality Attributes (QA), which are measurable or testable properties of a system that are used to indicate how well the system satisfies the needs of its stakeholders [3]. As we determined previously, sustainability has become an important non-functional requirement for software and, therefore, it needs to be taken into account during the design of the software, i.e. the system design phase in Figure 3.3. The reason for this is that, according to [3], the satisfaction of a software system's "qualitative" requirements heavily depends on the design and quality of its software architecture (SA) and, in return, the quality of the software design depends highly on the mapping of requirements into its architecture, i.e. the quality of the software is determined by the degree to which the architecture satisfies the requirements [24].

The preceding paragraph argues that the non-functional requirements and SA (e.g. its quality and design) are strongly related. There are many definitions for the concept of SA, but one of the better known definitions entails the set of (high-level) structures needed to reason about a software system, which comprise the software elements, the externally visible properties of those elements, and the relationships among them [3]. One of the main purposes of designing a SA is to gain insight in the qualities of a system (e.g. to what extend requirements are fulfilled) at the earliest possible stage. Therefore, an important role of the software architect is to balance between different QAs of a system so that they are best aligned and enable software with maximum quality. In support of this role, different models as well as standards related to QAs have been introduced, such as the ISO quality standards for measuring architectures by [32] or the Common Quality Attributes by [37]. The latter is depicted in Figure 3.4 and includes QAs on functionality, reliability, usability, efficiency, maintainability and portability. In addition, there are other research efforts which aim to support the implementation of these standards, such as: the classification and comparison of SA evaluation methods (e.g. [64]) and also the process of software architectures analysis (e.g. [11, 56]).

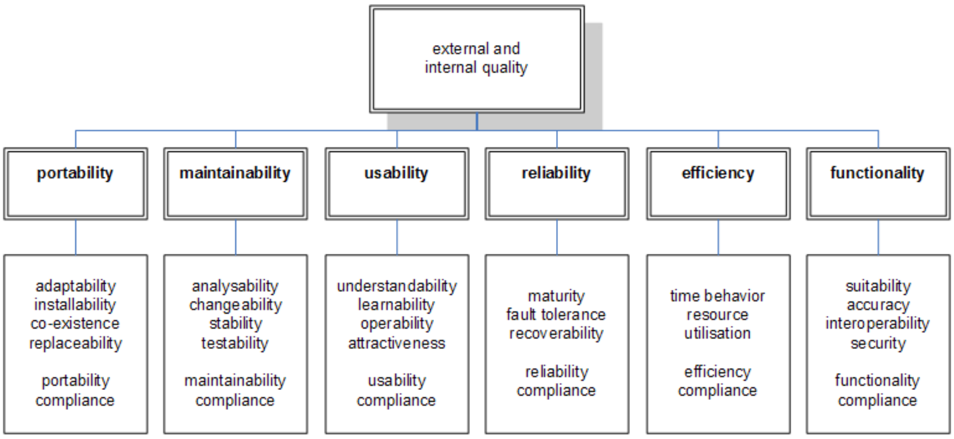


Figure 3.4: Quality model for internal and external quality by [37]

Currently, there is not yet a standardized QA for sustainability, as we can see in the model depicted in Figure 3.4. However, in the case that someone wants to develop a new QA (e.g. for sustainability), [3] includes the following procedure that can be applied. The first thing to do is interview the stakeholder whose concerns have led to the need for the QA. By working with them, either individually or as a group, a set of attribute characterizations can be build. Then, based on the resulting set, *scenarios* can be developed that characterize what is meant by the QA. According to [3], a QA scenario consists of the following parts: a stimulus (e.g. initiate x transactions), stimulus source (e.g. users), response (e.g. process), response measure (e.g. within y seconds), environment (e.g. normal operation), and an artifact (e.g. the system). After a set of guiding QA scenarios is constructed, a set of design approaches needs to be assembled for dealing with the scenarios. In support of this assembly, an architect can often make use of a collection of primitive design techniques i.e. *tactics*, which focus on achieving a specific QA. Thus, by effectively applying the right (set of) architectural tactic(s), a specific QA requirement is realized more easily.

Chapter 4

Current Practices

This chapter comprises the current practices with respect to software sustainability. The current practices are determined by conducting a literature and a company study, i.e.: a systematic literature review and a brainstorm session with domain experts, respectively. Consequently, this chapter contains a separate section for each study, including descriptions of the design, results, and analysis. The initial aim of both studies is to combine the concepts and theories introduced in Chapter 3 by focusing on the potential role of software architecture in realizing greener software.

4.1 Literature on Software Sustainability

A systematic literature review (SLR) was conducted to identify, evaluate, and synthesize the existing body of completed and recorded work produced by researchers, scholars, and practitioners [43]. According to [19], conducting a SLR has multiple purposes in a graduate thesis: (1) it synthesizes the understanding a student has on their particular subject matter, (2) it stands as a testament to the student's rigorous research dedication, (3) it justifies future research (including the thesis itself), and (4) it welcomes the student into scholarly tradition and etiquette. To ensure that we applied a "systematic, explicit, comprehensive, and reproducible method", our approach follows the eight-step guide of [43] for conducting a SLR of information science research.

4.1.1 Review purpose

The primary purpose of the SLR is to create a substantial theoretical foundation for the remainder of this thesis. In this thesis we attempt to bridge the gap between two research fields, i.e., Software Architecture and Green Software. Of these two research fields, the latter was found to be in an *emerging* state and the former in a much more *mature* state. Initially, the SLR was targeted on research that combines both these research fields, as showed in the following subsections. However, it turned out that the combination of these two research fields is still rare. Therefore, the SLR was extended by identifying available & significant research with respect to Green Software in general.

4.1.2 Systematic Literature Review Approach

In accordance with the guide proposed by [43], the approach of this SLR consists of 8 consecutive steps that are divided into four categories: Planning, Selection, Extraction, and Execution. Figure 4.1 displays a graphical representation of the relations between these steps and categories.

As the previous subsection already comprises the first step, i.e. *SLR Purpose*, and the second step, i.e. *Protocol and Training*, can be skipped since the SLR is executed by one reviewer (cf. [43]), further explanation of the Planning category can be omitted. Consequently, we can continue with the activities of the Selection category: *searching for the literature* and *screening for inclusion.*

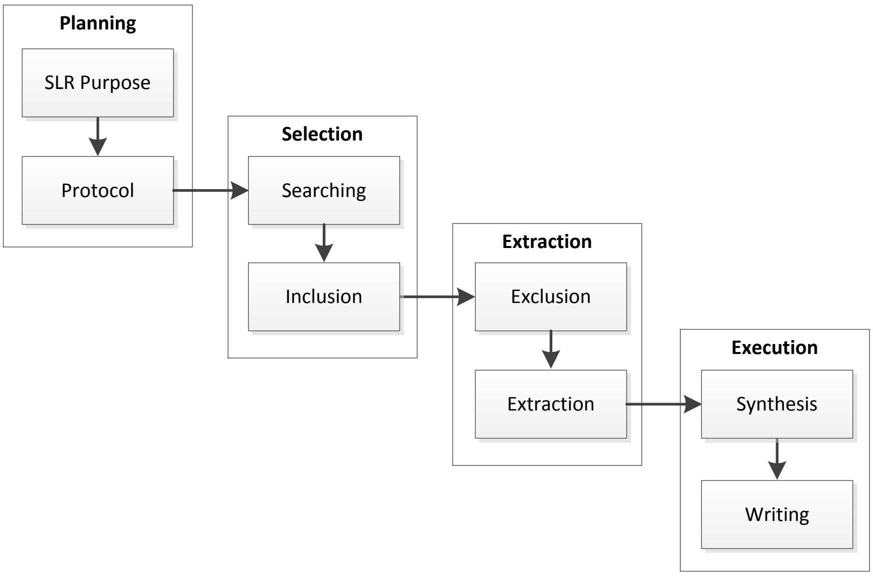


Figure 4.1: Structured Literature Review Approach

4.1.3 Searching for the Literature

Multiple digital libraries were considered as sources for the SLR: ACM Digital Library, IEEE Explore, Google Scholar, Microsoft Academic Search, SpringerLink, and ScienceDirect. We considered these libraries because they offer plenty of literature in the field of Information and Computing Sciences and, also, the library of Utrecht University offers (proxy) access to most of their material. However, when we began using their advanced search engines, we discovered that most of these libraries did not provide sufficient support (given our combination of search terms, see next subsection). For instance, Microsoft Academic Search has no clear syntax; IEEE Explore has a maximum number of search terms and (on second thought) not many accessible documents; ACM also has limited search options and SpringerLink has (or had) a broken search engine. Therefore, we concluded that ScienceDirect was the only digital library good enough for our SLR (see [9]).

4.1.4 Screening for Inclusion

Digital libraries, such as ScienceDirect, offer large collections of studies. Unfortunately, the vast majority of these studies are not relevant for a particular SLR. Thus, the next step in conducting an SLR is the practical screen, which entails the decision process of which studies should be considered for the review. More precisely, the purpose of the practical screen is to exclude unrelated studies based on (arbitrarily) chosen criteria so that the literature review remains practically manageable [14]. Furthermore, being explicit about these criteria ensures that the resulting literature is reproducible. Criteria that can be used to exclude studies are, among others, subject, content (type), keywords, publication language, authors, settings (e.g. domains), and date of publication.

Given that there are not yet many books on Green Software, we decided to limit the SLR to include only journals as content type. Other criteria were: no restrictions on the date of publication and authors, English as the publication language, and both the science domains of *Computer Science* and *Business, Management & Accounting* were considered. Next, we determined a set of search terms that would narrow the number of studies significantly. These terms are based on the initial research questions and include "synonyms" of the following 4 terms: Green Software, Software Architecture, Metric, and (Business) Goal. Figure 4.2 displays the resulting search terms. These synonyms were used to cope with different naming conventions (e.g., emerging research field) and to prevent that the search scope would also not become too limited. However, logical operators were used in such a manner to ensure that only combinations of the 4 initial terms or one of their synonyms would be included. This was achieved as follows: the terms in a single column, or group, are linked with each other by "OR" operators. This ensures that an article must satisfy at least one term in a column. In addition, the 4 columns are linked with each other by "AND" operators (e.g. the star symbol). This ensures that the satisfaction of at least one term in a column must hold for every column. The complete search term string can be found in Appendix C.1. Although this approach resulted in a long and complex search string, it enabled us to produce all the results in a single search. This resulted in a significantly shorter search and process time, because a fewer number of actions needed to be performed and fewer numbers of search records needed to be examined (i.e., far less duplicate search results were produced).

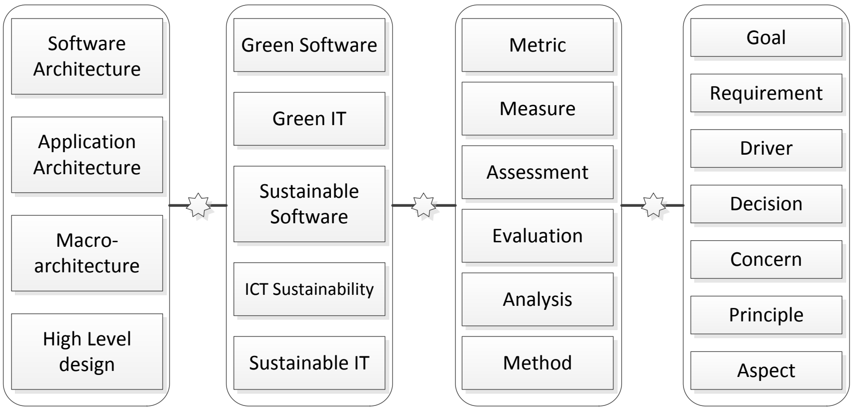


Figure 4.2: Structured Literature Review Search Terms

NB. Besides that ScienceDirect is the only digital library that supports such a complex search term, it also provides the best method for bulk exporting of search results. To make the screening process more manageable and reproducible, we settled with only 1 library, but at least a good one.

NB. By using a complex search term, our search resulted in 296 articles, which is a relatively low number of articles. Reasons for this may be that research which combines Software Architecture and Sustainability is still rare, or it can be that the the search term is too specific.

4.1.5 Exclusion Criteria

After we conducted the search and exported the result into a suitable format, the 296 articles had to be scored for their quality, i.e. the process of screening for exclusion. This process was performed in 3 consecutive steps, as shown in Figure 4.3. In the first step, the articles were judged based on their titles. An article was discarded if (1) none of the terms were present in the title, (2) the title implied a strong focus on Green IT (i.e., hardware, cloud computing), or (3) if the article mainly focused on a particular country or business domain (other than software development).

The result of the first step, or "filter", was that only 35 of the 296 articles remained. Then, in the second step, we read the complete abstracts of the articles and, subsequently, determined which of the articles would be relevant. This resulted in that an additional 21 articles were discarded and, thus, 14 remained. During the third step we read the content of these 14 articles and concluded that only 6 articles were both relevant and of sufficient quality. An overview, including detailed information of these articles, can be found in Appendix C.

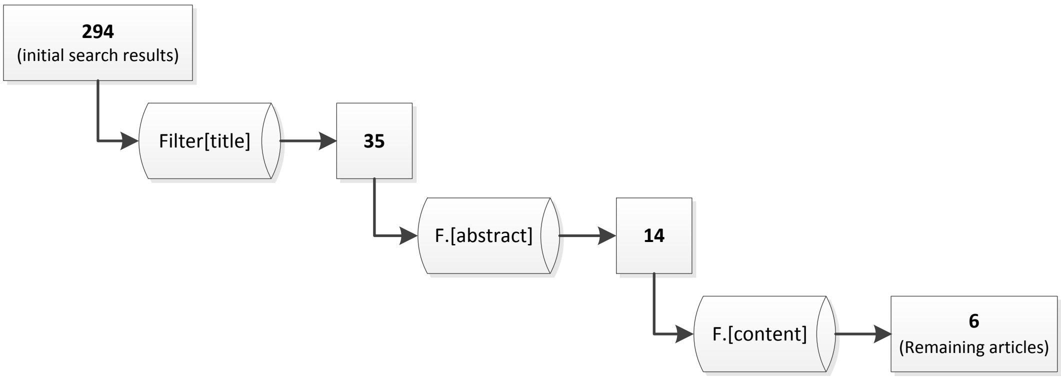


Figure 4.3: Structured Literature Review Search Terms

4.1.6 Need for Additional Literature

As the results show, the conducted SLR did not provide a large collection of new and relevant literature. In fact, 3 of the 6 articles were already discovered during preceding orientational literature searches. Therefore, we extended the collection of articles by including previous collected articles and performing additional (less systematical) searches, at other digital libraries and with other search terms. Furthermore, we applied techniques such as backward and forward reference searching (e.g. "snowballing") to retrieve significantly more articles. Eventually, this resulted in at least 50 useful articles and, also, several books, collections and conference proceedings. With respect to the latter, we saw a significant increase green software related conferences including, amongst others, ICT for Sustainability, or "ICT4S", and GREENS, as part of the ICSE conference. However, still only a few of the articles dealt with both Software Architecture and Green Software.

4.1.7 SLR Conclusions

The conclusion of this SLR is similar to the studies performed by [45, 5]: the collection of literature, which combines the topics of software architecture, metrics or aspects, and green, or sustainable, software, is small. However, the SLR was conducted more than a year ago, i.e. January 2014, and we have observed that the collection has grown since then. For instance, at the beginning of March 2015, the same search string was applied on the search engine of ScienceDirect. This resulted in an additional 169 articles, of which 50 were published in 2015 and 119 in 2014. Although the relevance of these additional articles has not been verified extensively, it does appear that the size of the unfiltered search results has grown by more than 50% in a single year. Furthermore, with respect to the quality of the search string, we realize that the extensive combination of search terms was most likely too specific and that its complexity also seriously limited the number of digital libraries that we could use, because it was only supported by one of the libraries.

In addition, over the course of time, the scope of the research in this thesis has shifted from a theoretical-level to a more pragmatic-level. Therefore, the role of, for instance, the software architectural aspects play a less significant role than initially planned. Consequently, the role of the conducted SLR itself has also become less significant within the thesis.

Conclusively, due to the small amount of articles, which resulted from this SLR, and the believe that the collection of articles is not sufficiently representable (anymore), a separate section on the SLR's analysis and results is omitted. Nevertheless, we argue that conducting the SLR resulted in useful insights and that, eventually, sufficient literature has been collected. In addition, we believe that other researchers can benefit from the extensive description of the applied SLR approach.

4.2 Architects on Software Sustainability

We concluded from the SLR, that the research field of software sustainability is still young and that it is slowly gaining scientific substance. However, there are practically no methods available that are designed for achieving more sustainable software by means of software architectural aspects. Moreover, we assume that sustainability is yet far from being a software quality attribute for architects and developers, let alone that there is sufficient knowledge regarding energy consumption of software. Given these conditions, we decided to conduct a brainstorm session to define the actual problem (e.g., the unfamiliarity with sustainability of software, and how it can be influenced by the SA), and find, through creative thinking, the best group decision for a plan of action to solve it [44].

This brainstorm session was organized as follows. A single session of 2 hours was held at the research company, on March 19, 2014. Besides the presence of the 3 researchers *EJ*, *LB*, and *RS*, there were 3 other participants, i.e. software architects from the research company: *EH*, *KL*, and *NV*. The brainstorm session was led by the facilitator, i.e. *LB*. The role of the facilitator is to draw out the suggestions from the participants; not to impose her or his own opinions, while still using leadership skills to maintain the order and purpose of the session. Initially, we envisioned to follow the procedure proposed by [3] to come to architectural tactics and scenario's for a new quality attribute. However, this idea soon turned out to be not implementable, due to the unfamiliarity of the participants with the theoretical concepts. In addition, a more unstructured approach appeared to be more effective at the current stage. The summary of the session is presented below.

Researcher *LB* started the session and introduced the following question: *"How would you, as an architect or designer of a software application, respond if you would get the additional (non-functional) requirement that the specific application needs to be more energy efficient?"*

First, Architect *KL* responded by mentioning previous research which concluded, amongst others, that there is not a large difference in the energy use of a system between its low and high workload states. Therefore, you should always attempt to minimize the number of different systems that are used for an application. Moreover, by minimizing the amount of resources required for an application, you can increase the number of applications on a single system (e.g., the number of clients that a single server can process). However, this is only applicable if you are allowed (and able) to execute multiple instances of one or more applications, on a single system.

Next, Architect *EH* explained that in general the functional requirements are leading during the initial stages of development and that most often the non-functional requirements are implemented after the realization of the functional requirements. Participant *NV* added that this order makes it harder to realize certain non-functional requirements, because they are not sufficiently taken into account during the initial design stage. Subsequently, *EH* mentioned several architectural designs which are believed to have a energy efficient character, such as cloud-computing. However, not much is yet known about their actual energy usage benefits.

In response to a statement of *EH*, i.e. "nothing is (yet) being measured with respect to software sustainability", *EJ* presented the following question: "what should we measure and how can we achieve this?". Subsequently, *EH* responded that, in analogy with software *profiling*, we can execute an application both multiple times and over a longer period of time, while in the meantime we measure the energy consumption. The benefit of this method is that, with the right tooling (i.e., both suitable and reliable), we can collect sufficient stochastic data on which we can perform statistical analyzes. In addition, the difference between white-box and black-box testing is explained: within the context of software profiling, developers have the possibility to focus on specific parts of software (i.e. a *white* or *clear* box), or a complete system (i.e., *black-box*), which is the only feasible option with most energy consumption meters. Thus, how should we cope with this lower granularity if we want to measure the energy consumption of an application? In response to this obstacle, *KL* stated that at least within a Windows platform, we can use performance counters to determine the amount of systems resources an application requires, such as the amount of processing time or committed memory bytes. Hence, the following idea emerged: we can monitor these performance counters during an execution of the application and determine the portion of system resources occupied by the application. Subsequently, we can use this to deduct the portion of energy consumption related to the application from the total amount.

Now that a strategy was determined for how and what we would measure, the session continued with the role of the software architecture: which architectural aspects could we investigate that might have an influence (be it positive or negative) on the sustainability of software. A brief discussion followed with examples from within and outside the context of the research company. Eventually the discussion resulted in a categorized list of architectural aspects that potentially have tangible influences on the energy consumption of software, i.e.,

- Software structure
  - Complexity (e.g., multi-core processing, dependencies, caching, etc.)
  - Pattern (e.g., Layered, MVC, C&C, SOA, etc.)
  - Scheduling (e.g., events, calls, processing)
- Software deployment
  - Multi-tier
  - Multi-tenancy
- Software frameworks
  - Interpreter (e.g., with or without)
  - Libraries (e.g., dynamic vs. static linking)

Subsequently, the participants were asked to come up with ideas on how we could measure the influence of at least one of the mentioned architectural aspects. With respect to the software frameworks, one answer entailed comparing different frameworks with similar purposes. For instance, *KL* mentioned that there are different methods for **dependency injection**, which is a specific design pattern, but often developers do not know which method should be used within a certain situation. To study this problem, we would need to build another (or adapt an existing) application so that there are at least two different versions having separate frameworks, we can determine whether the frameworks have different influences on the energy consumption. Another idea was to compare different version of an application (e.g., an older vs. a newer version), and determine which architectural aspects have been modified between the two versions. Then, depending on the amount of modification or tactics, we could determine whether a (set of) tactic(s) has a positive or negative influence on the energy consumption.

Lastly, Architect *EH* mentioned that he was part of the development of *DOCGEN*, an application for document batch processing which supports **multi-core processing**. Since the software of the research company is mostly data oriented and that it is believed that the performance benefits are not significantly large within this category of software, multi-core processing is not yet a standard capability of the research partner's software. An important reason for this is that the implementation of multi-core processing requires additional development resources. Therefore, the benefits during the operation of the software should be significantly higher than the additional development costs. Consequently, the idea was formed that it would be interesting to investigate to what extend the amount of operational resources required by the multi-core version differs from the single-core version, and how this might influence the energy consumption of the system.

The brainstorm session was concluded by briefly summarizing the things discussed during the session, inform the participations of our plan of action and thanking them for their contributions.

4.3 Synthesizing the Current Practices

By combining the results from both studies, we concluded that (1) the research field of software architecture and green software is small, but increasing, and (2) the practical knowledge is even more scarce. Nevertheless, the studies did produce a comprehensive overview of available literature as well as multiple options to conduct experiments with. Consequently, before we will introduce our final hypotheses (see Chapter 5), we discuss some of the preceding considerations in this section.

The first consideration concerned our approach for comparing different versions of an application to be able to determine the influence of architectural aspects. Inspired by the method for developing a new quality attribute by [3], we initially considered selecting two different versions of a single application, of which the latter is a successor of the first. As is depicted in Figure 4.4, we could then determine which tactics were applied in each version and, subsequently, measure whether the tactic(s) implemented in the latter version resulted in a positive or negative influence on the sustainability of the application. This way, we would not have to develop the software, which would be a significant benefit due to a limited amount of time available, and we could still determine whether a tactic has a negative or positive impact (e.g. through "reverse engineering"). However, there were some limitations to this setup. First, it would only be possible if we could find a suitable application (e.g. multiple versions with a clear overview of tactics per version). Second, most applications contain multiple tactics per version and, therefore, it can be hard to determine the exact influence of a single tactic. Third, if we could isolate the influence of a single tactic, we would still need to validate its influence within a different environment. Since we did not succeed in finding a suitable application, we eventually decided not to focus on comparing incremental versions. Instead, we are considering "configurations" of an application as versions.

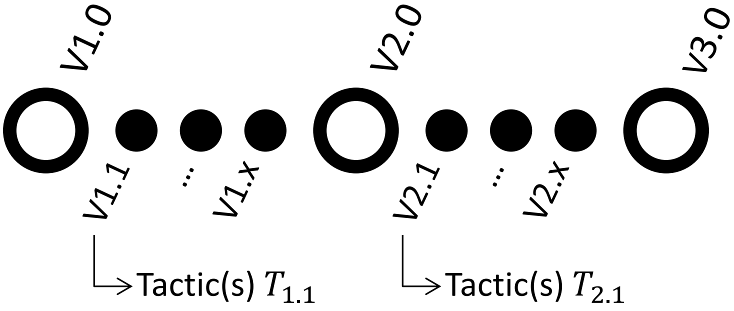


Figure 4.4: Representation of Incremental Application Versions

The second consideration concerned the selection of a software architectural aspect from the list of candidate aspects. The main two candidates considered were dependency injection (DI) and multi-core-processing (MCP). The first was considered because one of the brainstorm participants mentioned that, at the moment, many developers and architects have no clear preference for a certain DI implementation, or *container* [15]. Furthermore, an extensive search for a useful test application resulted in a complete environment in which we could easily select a different DI container. In addition, this environment was also written in *vb.net*, a programming language which is broadly applied at the research company. Finally, the main function of the test application was to calculate Pi decimals as fast as possible. This is, of course, an intensive task that mostly uses a system's central processing unit (CPU) component. We quickly discovered that such a task, or "benchmark", would be best to focus on, because it appeared to produce the most clear workload profile (e.g. single component and relatively constant). However, after analyzing the DI option more in depth, we concluded that the differences between DI implementations (mostly) occur during the compilation of the software and, therefore, we assumed that the (potential) differences with respect to the operational energy use of the software would be insufficiently significant.

Then, the second aspect, i.e. MCP, was considered because of the opportunity to experiment with an application developed by the research company, namely *DOCGEN*. Since MCP is not often implemented within the related type of software, it would be interesting to investigate whether its potential sustainability benefits could have a significant influence on the decision making process. However, before these benefits can be determined, we would first need to be able to measure the energy use of different implementations. Since this would be relatively difficult with *DOCGEN* (e.g. an application with a more complex design and deployment scheme than we preferred at that stage), we decided to perform an initial experiment with a "simple" application before we would consider *DOCGEN*. Therefore, we searched for an additional application that provided multiple configurations with respect to MCP (e.g. to be able to compare different versions) and, also, one that produces a clear workload profile (i.e. a system benchmark application). This search resulted in the following candidates: *primesieve* and *wPrime*, which are both prime number generators, and *Hyper PI*, which is a multi-threaded version of *Super PI* (e.g. one of the better known benchmark applications). However, these candidates were all rejected eventually because they did not provide the option to operate the application with specific parameters from command-line. This was necessary because we determined that we needed to configure and execute the applications automatically and remotely by means of a batch script. Luckily, we eventually did find a suitable application that supports all our requirements, namely *Systester* [60]. In Section 6.1, we will describe additional details of this application including its various configuration options.

The third consideration concerned finding the right instrumentation that would be used during the experimentation. First of all, we had to be able to measure the energy use of the application and/or system. For this, we initially envisioned to use *Joulemeter*, an application for estimating the power usage of systems that, in addition, has the option to isolate the energy use related to a single application [18]. However, despite that it is one of the better so-called "energy profilers" currently available [26, 2], we eventually concluded that *Joulemeter* has too many limitations for our experimentation. For instance, the power model used by the application can only be calibrated on a laptop (e.g. through the discharging of the battery) or via an external *WattsUP?* power meter. Furthermore, this application does not enable the measuring of energy use remotely and, also, the reliability of Joulemeter did not appear to be sufficient for us during orientation measurements. The previously mentioned WattsUP? power meter (WUP) did not have these limitations and, luckily, we were able to obtain one (e.g. receiving the correct model was not a simple task).

Figure 4.5: Instrumentation: a WattsUP? PRO power meter

In addition to the measuring of energy consumption, we also needed instrumentation for monitoring the workload of the system. For this we considered multiple options, mostly due to differences in detail, reliability and ability to export per option and, also, the ability to remotely monitor a system. For instance, we considered the Windows build-in Performance Monitor, or *PerfMon*, and *Xperf*, which is part of the Microsoft Windows Performance Toolkit. We considered the latter because it can produce significantly more detailed monitor logs, while still reaming a very low overhead, and it also provides the option to record on a higher granularity than PerfMon, which can only create 1 recording per second. However, Xperf does not provide the option to remotely monitor a system and it is also more difficult to customize a data collector. Both these things are included in PerfMon and therefore we preferred PerfMon over Xperf, despite that the connection with a remote system can be troublesome and the recorded data is less reliable (due to the lower granularity of the tool). In addition, we attempted to overcome the loss in the detail by implementing the Intel Performance Counter Monitor (PCM), which can be used on itself or as a plug-in for PerfMon [65]. By using PCM, we would be able to monitor the exact number of procedure calls, or "clock ticks", per second and, thus, we could attempt to relate the energy use to the actual amount of work processed by the CPU. This would still be per second, since both the WUP meter as PerfMon have a granularity of at most 1 recording per second, but the data would be more detailed since PerfMon alone is not able to record the actual frequencies of the CPU(s) of a system.

Although the implementation of PCM within PerfMon provided multiple technical obstacles, we were eventually able to use it from a remote system. However, it soon appeared that the combination of remotely monitoring and using PCM as a plug-in in PerfMon resulted in recordings with many gaps. In addition, PCM is supported by only a limited collection of Intel processors and since one of the test system was not part of this collection, we eventually decided not to use PCM and, thus, to settle with a standard version of PerfMon. However, by doing so, we would still need to take an additional action to overcome the limitation of PerfMon with respect to not being able to record the exact amount clock ticks per second. The reason for this, is that many systems have varying CPU frequencies (e.g. to match the estimated workload and save energy), but for the experiments we need to be able to keep the frequencies fixed, otherwise it would be more difficult to compare measurements. Although locking the frequencies of a system's CPU(s) can sometimes be achieved trough its *BIOS*, this option was not supported by every test system. So, to standardize our method among the systems, we used an application to overcome this, namely *Throttlestop* [17].

Finally, besides being to able to lock the CPU frequencies, we also needed to determine which values would be best for the whole group of systems. During orientational measurements we observed that the CPU frequency has a strong influence on the resulting power consumption of a systems and the required time to execute an application. For instance, by locking the frequency of one of the test systems to 800MHz, instead of 2300MHz, the execution time increased threefold while the power consumption decreased by 50%. Consequently, the total amount of energy the system required to execute the application at 800MHz was higher that at 2300MHz. Unfortunately, despite that all of the included systems have an unlocked CPU multiplier, there were several things that prohibited us from conducting such a comparison with each system. First, the ranges between the minimum and maximum CPU frequency of the different systems did not overlap sufficiently, e.g. the maximum of one system was the minimum of another system. Second, we realized that it would be more valuable to use similar frequency values when comparing between the systems, which was not possible due to the differences in ranges. Third, we observed during measurements with systems performing at their maximum CPU frequency that the power consumptions resulted were significantly less stable, due to a buildup of heat. Thus, given the available systems for testing (see Section 6.1.2), we concluded that it would be better to select a single CPU frequency, i.e. 2GHz, since this value is supported by each system and it does not result in a buildup of heat.

In summary, we have determined the following: (1) consider multi-core processing as a candidate architectural aspect for sustainability, (2) conduct experiments with a simple and a complex application, (3) use the WUP and PerfMon as instrumentation, and (4) lock the CPUs at 2GHz.

Chapter 5

Building Metrics for Sustainability

In this chapter we introduce the hypotheses that will be tested during the experimentation. But first, we re-discuss the context of the research briefly so that we can "prepare" the main aspects.

5.1 Recap of the Research Context

Within the field of software architecture, non-functional requirements, or quality attributes, are used to evaluate the quality of software [3]. Furthermore, a quality attribute should be *objective* and *quantifiable*, which entails that there must be some measurable way to assess to what extend the attribute has been realized. Consequently, to come to a quality attribute for the "sustainability" of software, we need reliable metrics that are both objective and quantifiable. Further, these metrics should allow for comparison between different applications or versions of applications.

Sustainability has a broad interpretation, ranging from the energy consumption by the hardware during execution of the system, as well as organizational and the complete life-cycle of the system, including development and maintenance. Consequently, deriving a complete and sound set of metrics is difficult, if not impracticable. Furthermore, there are different types of software: this research focuses on product application software [66]. As product software is typically installed at many different clients, the main driver for sustainability is a product's resource requirement during execution instead of, for instance, during development. Hence, we solely focus on its operation.

Seen from an execution point of view, energy consumption is the most logical measure to come to a reliable, i.e., objective and quantifiable, metric. Additionally, a software product might run on many different systems (e.g. new/old, laptop/PC). Therefore, ultimately, such a metric should be *independent of the system*, so that it is possible to predict the energy consumption of the software product upfront. So, in analogy with the field of construction engineering, we further focus on the *operational energy use* of applications, which entails the energy consumption during execution [50].

To come to a metric for the operational energy use of an application, which is thus independent of the system it is executed on, we need to overcome many challenges. For instance, a first obstacle would be limitations with regards to the instrumentation for measuring energy consumption, i.e. the WUP meter (introduced in Section 4.3) can only measure the energy use of the complete system on which an application is executed. Secondly, most computer systems have a variable energy use, partially caused by changing workloads such as background processes inflicted by e.g. the operating system. Thirdly, determining to what extend the system is processing the application is not straightforward, e.g. considering other (background) processes and limited instrumentation. Lastly, systems may have different energy consumption properties (e.g. ranging between idle and full workload) In the next section, we will show how we take these challenges into consideration to obtain a (more) reliable and independent measure for the operational energy use of applications.

5.2 Measuring the Operational Energy Use

Due to the presence of multiple challenges, we chose to conduct a staged experiment. In addition, to test the independence of the metric, we performed measurements with multiple application versions and different test systems. For each combinations of an application version and a system, i.e. an *AppSys*-configuration, we performed multiple measurement runs to minimize the amount of noise in the data (e.g. due to random background processes). Figure 5.1 depicts a representation of the activities related to performing multiple measurements with a single *AppSys*-configuration, i.e. a series. This representation shows that for each execution of the application, i.e. $E_1, E_2 ... E_n$, the instrumentation produces the related measurement data, i.e. $D_1, D_2 ... D_n$. Subsequently, the metric results, i.e. $R_1, R_2 ... R_n$, are calculated by applying a (set of) metric(s) on the measurement data, which can be related to a single execution of A or a series (e.g. by averaging over the runs).

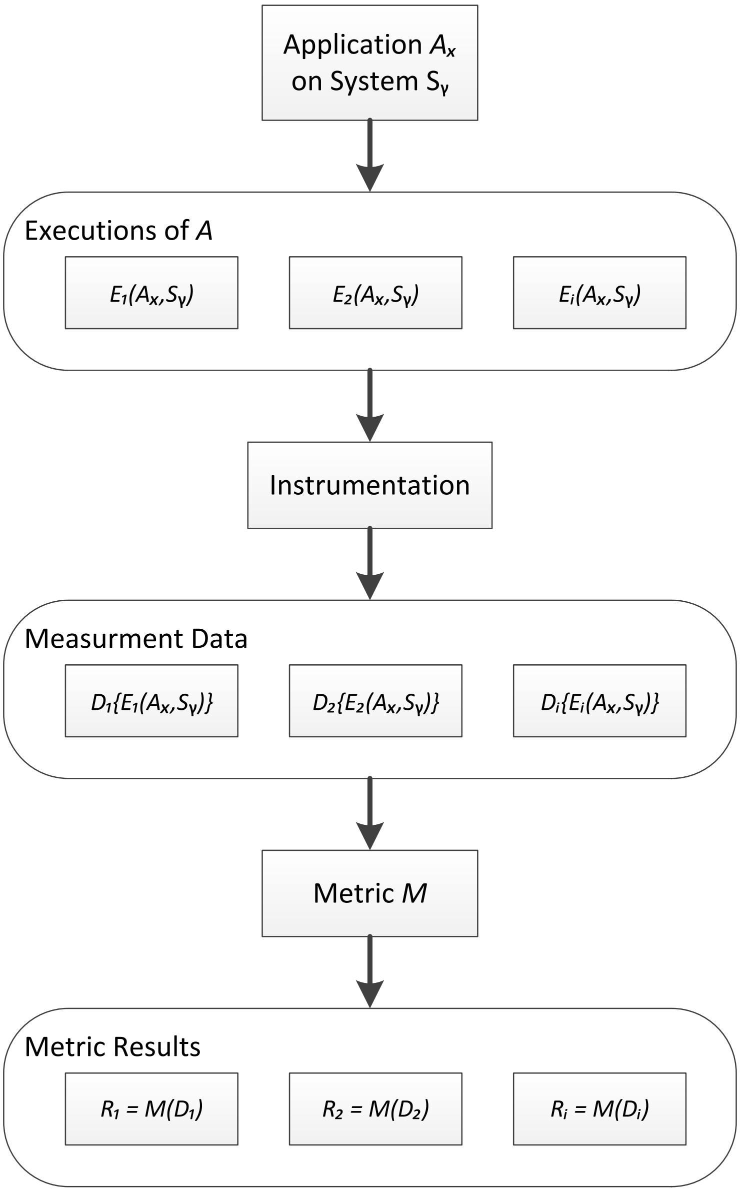


Figure 5.1: Representation of measuring multiple executions of A

Then, Figure 5.1 depicts a representation of how we conducted measurements with the different *AppSys*-configurations. This representation includes multiple configurations versions, i.e. $V_1, ...V_i$, as well as multiple (and different types of) executing systems, i.e. S_1, S_2, S_3. Consequently, the results per *AppSys*-configuration, i.e. $R(V_1, S_1)...R(V_1, S_3)$, $R(V_i, S_1)...R(V_i, S_3)$, are calculated by applying a (set of) metric(s) on the measurement data that result from their related executions.

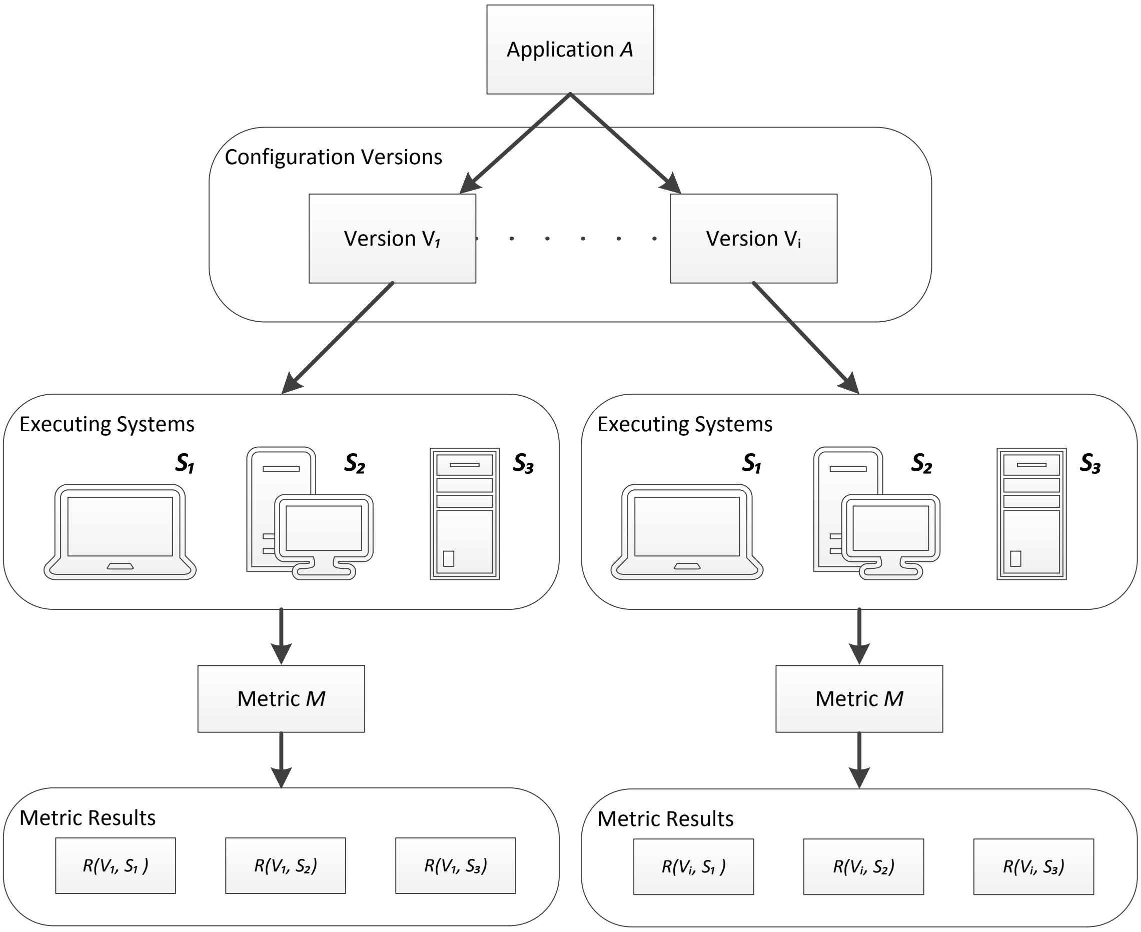


Figure 5.2: Representation of measuring with multiple *AppSys*-configurations

Given the representations depicted in Figure 5.1 and Figure 5.1, we aim to determine the independence of the metric by means of the following steps. First, by averaging the data and results over multiple measurements, we can neutralize the influence of noise that may occur during measurements. Furthermore, by using a constant *AppSys*-configuration, we can test the reliability of the metric in a single environment. Figure 5.3 gives an abstract representation of these two initial steps. Second, by conducting measurements with a single application version executed on different systems, we can further compare the metric results and determine to what extend there are differences (or similarities) between the systems and, possibly, determine the causes of these differences. Finally, by conducting measurements with different application versions executed on a single system, we can also more safely compare the results between different versions (e.g. to determine the influence of design decisions). The latter two steps, or comparisons, are illustrated by Figure 5.4 and Figure 5.5, respectively. As a final step, we combine the last two comparisons and determine whether differences between different versions remain constant over different systems. Consequently, these steps enable us to properly validate the stability and reliability of the metrics.

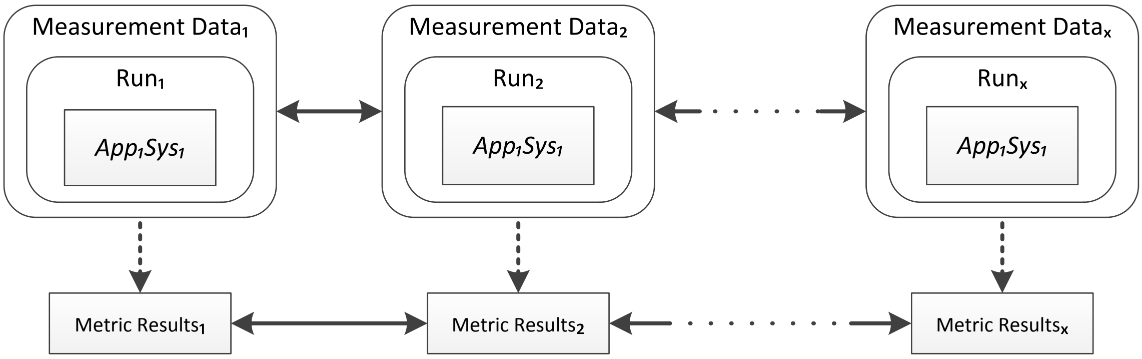


Figure 5.3: Comparing data (and results) from multiple runs with a single *AppSys*-configuration

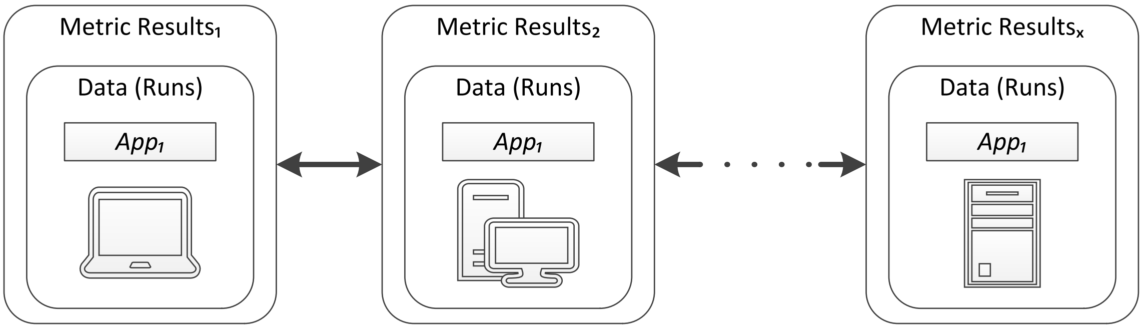


Figure 5.4: Comparing results between different systems with a single application version

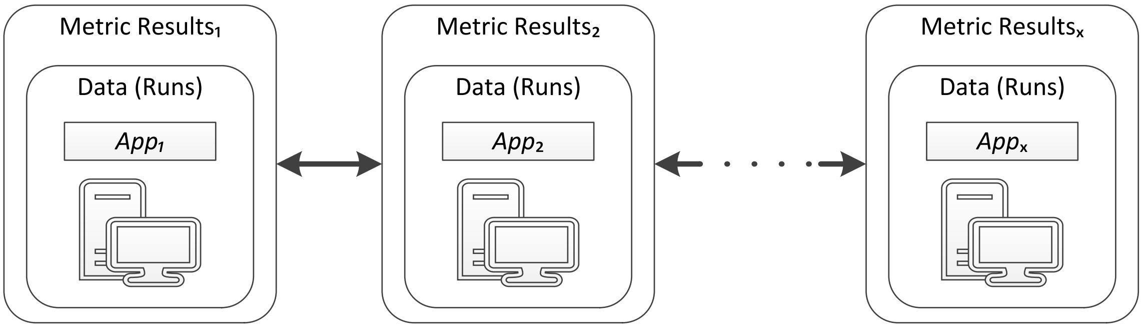


Figure 5.5: Comparing results between different application versions with a single system

5.2.1 Introduction of terms

As explained before, the WUP meter can only measure the total energy consumption of a system, i.e. S. Nevertheless, we aim to determine the amount of energy solely related to the execution of an application, i.e. A. Therefore, we study the ratio between the energy that is being consumed by S in its base state, i.e., the system running without executing any significantly "heavy" applications, and the energy consumed while solely executing A. We follow the SI standards for the notation of power $P(\cdot)$, i.e. Joules per second or Watts (W), and energy $E(\cdot)$, in Joules. Consequently, $P(\mathrm{S}^A)$ and $E(\mathrm{S}^A)$ refer to the average power and the total energy consumption of S during the execution of A, i.e. S^A, respectively. In addition, S^B and S^M refer to S while it is in its base state and when it has a "maximum" system workload, respectively. Finally, besides the measuring of energy use values, we also record the amount of time required for S^A, which we denote by $T(\mathrm{S}^A)$, and the performance of a system (e.g. the active processes and workload at components) during S^A.

5.2.2 Towards a Normalized Operational Energy Use

Based on the arguments presented above, we determined the following set of metrics. The first is to inspect the average power consumption of system S during the execution of application A, i.e.

$$P(S^A) = \frac{E(S^A)}{T(S^A)} \tag{5.1}$$

However, this metric is not yet independent of the system it is executed on. To make the metric more independent, we want to inspect the energy solely related to the execution of application A on system S, i.e. A_S, by subtracting the energy use of S related to its base state from $E(S^A)$, i.e.:

$$E(A_S) = E(S^A) - E(S^B) \tag{5.2}$$

However, as it is not possible to both measure $E(\mathrm{S}^B)$ and perform S^A concurrently, we assume that the amount of energy use related to the system's base state remains constant during S^A, i.e.,

$$E(A_S) = E(S^A) - P(S^B) \cdot T(S^A) \tag{5.3}$$

Although this measure eliminates the energy consumption related to the base state of the system, it does not allow us to compare the energy consumption on different systems, as the produced results are absolute and there can be significant differences in the scales of system power usages. For instance, let S_1 be a laptop system, S_2 be a standard personal computer, and S_3 a server. Then, the average power values of S_1 in its base and maximum state can be 10 W and 42 W, respectively, while for S_2 it can be 70 W and 90 W and for S_3 260 W and 285 W, respectively.

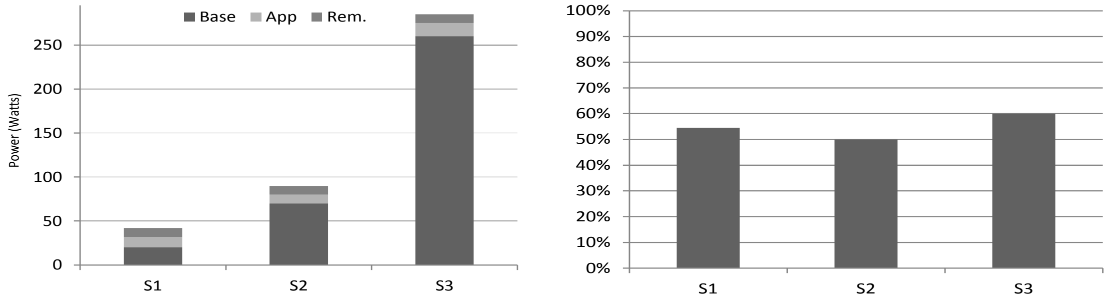


Figure 5.6: Example of power distributions per system :: absolute P [left] and $R(A_S)$ [right]

To illustrate this example, we included Figure 5.6, which contains two separate graphs. The left graph shows, besides the introduced base and (remaining) maximum power values, the amount of power solely related to S^A, which clearly illustrate the differences in scales between the systems.

One way to overcome these differences in scales, is to normalize the power consumptions. This can be achieved by means of *feature scaling*, which entails in our case subtracting the base power value of each system, i.e. $P(\mathrm{S}^B)$, from both its base and maximum power values, of which the latter occurs when a system has a full workload, i.e. S^M. Subsequently, these new values give us an interval, or *range*, in which we can relate the normalized power consumption during S^A, i.e. $P(\mathrm{S}^A) - P(\mathrm{S}^B)$, to the rescaled minimal, i.e. 0, and maximal power consumptions, i.e. $P(\mathrm{S}^R) = P(\mathrm{S}^M) - P(\mathrm{S}^B)$. Then, if we assume a linear dependence between a system's workload and the resulting $P(\mathrm{S}^R)$ values (e.g. 50% within a system's workload range ≡ 50% within its power use range), we can use the following metric that is less sensitive to different absolute power use scales:

$$R(A_S) = \frac{P(A_S)}{P(S^m) - P(S^B)} \tag{5.4}$$

This metric results in a ratio between zero and one, as is showed in Figure 5.6b, which contains the results of Metric 5.4 applied on the example values. This ratio gives an indication of how much S^A increases the power consumption of S in comparison to its $P(\mathrm{S}^B)$ and how much there is still left until it reaches $P(\mathrm{S}^M)$. For instance, an $R(A_S)$ value of 0,7 would indicate that on average S requires 70% additional power during S^A in comparison to its $P(\mathrm{S}^B)$ value. Ultimately, the $R(A_S)$ metric can support us in predicting the average power consumption of an application on a specific system, provided that the $R(A_S)$ value is only strongly dependent of known system characteristics, such as the number of CPU cores. Moreover, given that we can reliably determine the $P(\mathrm{S}^B)$ and $P(\mathrm{S}^M)$ values of a specific system (e.g. to have a correct *power use range*), an initial metric for predicting the average power consumption of S during S^A would be as follows:

$$P(S^A)* = R(A_S) \cdot (P(S^M) - P(S^B)) + P(S^B) \tag{5.5}$$

In addition, we can also consider predicting the required operational energy for S^A by means of the workload at the system or at a specific component, provided that there exists a significantly strong correlation between the workload and the resulting power consumption of S related to S^A. We denote the workload at the system during S^A by $L_S(S^A)$ and, given that we aim to focus on the CPU component (i.e. the main energy consumer [33, 13]), we would denote the workload at this component by $L_{CPU}(S^A)$. Then, if there exist a significantly strong correlation between the workload at the system, or a specific component, and the resulting power consumption, we can predict the latter by means of i.a. the former. However, if the correlation is significant but the values are not exactly similar, we need to use a conversion factor, i.e. f, which has the value 1 in the case that the values are exactly similar. The CPU focused version of this metric is as follows:

$$E(A_S)* = L_{CPU}(S^A) \cdot f \cdot P(S^B) \cdot T(S^A) \tag{5.6}$$

One of the important assumptions, that requires validation, is whether the power related to the base state of the system can be subtracted from its total power consumption. Therefore, we record all the processes executed by the system's CPU during S^A to check whether no other processes occupy a significant portion of the system's processing time while executing the application. In addition, we use this to determine the amount (and cause) of *noise* during the execution, and thus to decide whether measurements are sufficiently *clean*. Finally, we need to investigate to what extend the different variables are correlated, before we can actually make predictions of results.

Chapter 6

Experimentation

This chapter describes the experimentation that was performed during this thesis project. The main purpose of this experimentation is to investigate the hypotheses, e.g. the metrics, introduced in Section 5.2.2. To do so, Section 6.1 begins with the experimental setup, including the specific instrumentation, systems, test applications and experiment protocols. Further, the experimentation results will be presented in Section 6.2, followed by the analysis of these results in Section 6.3, and, finally, additional considerations with respect to the experimentation are discussed in Section 6.4.

6.1 Experimental Setup

As discussed before, the main goal of this research is to be able to determine the energy use solely related to the execution of application software. To achieve this, we introduced a set of metrics that will be tested by means of an experimentation. To ensure its quality and reproducibility, we determined an experimental setup that is based on the following rules. For the production of sufficient data, we executed and measured multiple *AppSys* instances, in a semi-controlled environment. Since the measurement data often varies between different executions (e.g. due to noise), we also executed each *AppSys* instance a multiple number of times. Further, we increased the quality of the measurement data, i.e. minimizing the amount of noise, by sampling over multiple repetitions and implementing control measures. With respect to the most optimal "number" and "duration" of repetitions: we conducted several orientation measurements to determine the maximum feasible values, which was a trade-off between quality and a limited amount of time available for measuring.

In addition, we only included systems with the following properties: each system has to run on Microsoft Windows 7 Professional or Windows Server and it must be equipped with an Intel dual-core processor. These two decisions were based partly on the most appealing subset of the limited number of available systems, but also both decisions provided us with several useful capabilities, such as *remote access* and *advanced performance monitoring*. Other selection criteria regarding system properties were not that strict, in fact: we included both a laptop, PC and a server (Nb. the energy use of the display was excluded), processors from different generations with different specifications, and different types of other components, such as the memory. This decision may be controversial, since it makes it more difficult to create a controlled test environment in which results between systems can be easily compared. Nevertheless, we determined that it would be impossible to create a perfect environment (with the given resources). Furthermore, we believe that (currently) it is more valuable to experiment with dissimilar systems, because in practice it is more likely that a software product will be executed on many different systems. Therefore, we assume that it is more meaningful to create research results that are based on different computer systems. Moreover, there will always be a remaining margin of error, since the ultimate aim of this research is towards being able to estimate, given a limited set of measurement instruments, an amount of energy necessary to execute a certain application independent of the executing system.

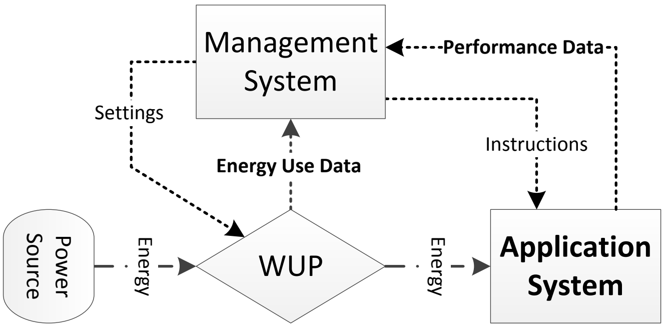


Figure 6.1: Representation of Experimental Setup I

6.1.1 Instrumentation

As is discussed in Section 4.3, we initially envisioned to use *Joulemeter* for estimating the power usage of systems. However, this software-based instrument turned out to be too unstable, amongst others. Therefore, we decided to use the more reliable WUP plug-load meter. As is depicted in Figure 6.1, the WUP meter is placed between an electronic device and the external power source of the device. It can register the power consumption of a device at a minimal interval of 1 time per second with an accuracy of +/- 1,5 percent [23]. Furthermore, it has the capability to measure up to 18 different electrical properties, such as *Power Factor* and *Line Voltage*. However, we only measured *Watts* and *Amps* to maximize the limited storage capabilities of the device. The benefit of the "Pro" version is that you can download the recorded data via USB to another computer (i.e. the *Management System*) and, in addition, modify the settings of the WUP meter remotely. An important benefit of this feature is that it excludes the *Application System* from the recording of its own energy use and, consequently, potential interferences during the recordings are minimized.

Besides the logging of the power usage data, we also monitored and logged the performance of systems during measurement. This was necessary because: due to the nature of the computer systems (e.g. existence of random and variable background processes), we had to be able to check (afterwards) whether a system was sufficiently idle during a "base" measurement and whether the system was (mostly) only processing the specific application task, during "application" measurements. Since all systems run on Windows, we were able to use the build-in Performance Monitor (PerfMon). PerfMon is a visualization tool for viewing and recording system performance data, which can be exported in a user-friendly CSV format. The tool provides the options to monitor performance counters, event trace data, and configuration information, which are combined into so-called "Data Collector Sets". For this research, we only focused on the performance counters, which are "measurements of system state or activity" [36]. Moreover, we created multiple customized Data Collector Sets, which included, among others, the *%Idle Time* and *%CPU Time* per CPU, and *%CPU Time* per process. By using the same IP address for the system that was being measured, we were able to use some of these sets on multiple systems, which made the comparison between resulting logs easier. In addition, as Appendix A shows, additional (e.g. non-CPU related) counters were included, such as memory, physical disk, network and the complete system.

6.1.2 Systems

The measurements were performed on three different Application Systems. The specifications of these systems are showed in Table 6.1. Each system represents a different computer class, i.e. a laptop, a desktop PC, and a server system. The main reason for including these systems, besides that they were available, is that they all have an Intel processor, with two available cores and an unlocked CPU multiplier (e.g. for locking CPU on a specific frequency), and no dedicated graphics card. It should be noted that the specific CPU of the *SRVR* system has in fact 4 separate cores (e.g. quad-core). However, its underlying infrastructure, i.e. *VMware ESXi 5.0*, restricts it to 2 separate cores performing at 2GHz. Furthermore, besides these systems, additional systems were considered, such as: a desktop computer with a single-core Intel Pentium 4 processor and 2 modern dual-core laptops with Hyper-Threading technology. These systems were eventually not included due to their number of CPU threads, e.g. 1 and 4, respectively. Also, the Pentium 4 did not have an unlocked CPU multiplier and, thus, its frequency (i.e. 3GHz) could not be adjusted.

Table 6.1: Overview of system specifications

	System		
Property	*LPTP*	*DKTP*	*SRVR*
Brand/model	ASUS F3JA	*Custom PC*	HP DL380
Processor	C2D T7200	C2D E6750	Xeon E5335
FSB & TDP	667 / 34	1333/ 65	1333 / 80
Chipset	Intel i945PM	Intel P35	Intel 5000P
Memory	2GB DDR2	2GB DDR2	4GB EDO
OS - SP/bits	W7P 2/32	W7P 2/32	W2K8 1/64

6.1.3 Application Versions

Given the purpose of the experimentation, we searched for a test application, which includes a “task” that would produce a clear and reproducible workload during its execution. The reproducible workload was necessary because we saw that systems often have random changes in terms of their workload and energy use, and by collecting multiple instances of measurements we would be able to filter out most of the noise. In addition, we also searched for an application that mainly uses the CPU component because in general it is the most consuming component, according to [33], and this way it would be easier to analyze and compare the data (due to the collection of performance counters available). After considering multiple options (see Section 4.3), we selected a system stressing and benchmarking tool named *Systester* (version 1.5.1) [60]. The main purpose of *Systester* is to test a system’s stability by calculating up to 128 millions of Pi (decimal) digits. The task of calculating Pi decimals is common for such system benchmarking tools to implement, since it requires large amounts of resources (e.g. CPU time) to calculate the larger decimal numbers and, therefore, pushes a system to its limits. Furthermore, the duration to finish such a task is relatively stable over repetitive executions, since the application is designed to minimize the calculation time by using as much of the system’s available computation power as possible. Consequently, the resulting calculation times can be compared with results from other systems, to see which system has better performance and stability properties. However, for this research we were not interested in producing the lowest calculation times, yet we were interested in *Systester*’s ability to (1) execute configurable tasks, which (2) keep a system busy for roughly 5 minutes, (3) produce mainly a high CPU workload and (4) require similar resources between multiple execution repetitions.

With respect to the most suitable duration of the task, we determined that the quality of the energy use measurement data (e.g. amount of noise), directly correlates with the duration of the measurements. The reason for this is that (in general) the impact of random but brief workload-spikes on the average energy consumption of systems decreases when the duration of measurements increases. Consequently, the quality of measurement data would be the “cleanest” if the largest task offered by *Systester* is selected, i.e., calculating 128 million Pi decimals. However, a downside of this would be that the number of measurements would be smaller due to a limited amount of time available for measuring. Therefore, after multiple experimental runs, we decided to select a task that would take around 5 minutes for most of the systems. In addition, *Systester* offers the option to select two different algorithms, namely: *the Quadratic Convergence of Borwein* and *Gauss-Legendre*. The latter is a significantly faster algorithm and is also being used other (more popular) benchmarking applications (e.g. *SuperPI* [61]). However, only the *Borwein* implementation supports multi-core processing. Therefore, we included both algorithms in our experiments. Finally, these considerations resulted in 3 application *versions*, which are included in Table 6.2.

Table 6.2: Overview of application versions

Code	Algorithm	# Decimals	# CPUs
Gaus8M1C	Gauss Leg.	8 x 10^6	1
Borw8M1C	Borwein	8 x 10^6	1
Borw8M2C	Borwein	8 x 10^6	2

6.1.4 Experiment Protocols

This section includes descriptions of the main steps that were made before, during and after the measurements. They are included to provide additional insight to the readers as well as the opportunity for other researchers to reproduce the measurements, more easily.

Preparing an Application System

Before a series of measurements can be performed, the Application System (AS) and Management System (MS) need to be prepared properly. To do so, the following steps need to be taken.

1. Place the WUP meter between the AS and the power source and connect it with the MS.
2. Copy the files of *Systester* to the AS. We used the Command Line (CLI) version of *Systester* which enabled us to execute the application from a Batch script with the desired parameters. In addition, we compiled a modified version of *Systester*, in which the application waits 5 seconds both after initiation and before ending of the process. This way, the PerfMon log were more reliable and it was easier to determine the interval of each specific execution run.
3. Disable obsolete Services on the AS, such as Windows *Back-up*, *Defender* and *Update*. Also, disable connection with the Internet and connect AS with MS through a wired connection.
4. Ensure that PerfMon (operated from MS) has access to the performance counters on AS.
5. Ensure that the CPU of AS is locked on 2GHz and that Throttlestop starts after a reboot.
6. Ensure that the AS can be rebooted remotely (e.g. without login screen) via PsTools [49].
7. Check the power settings of AS for abnormalities, such as hibernation and hard drive settings.
8. If AS is a laptop, remove the battery and turn off (or dim) the screen to minimize interference.

Preparing the Management System

When the AS is ready for a measurement, the following preparations need to be taken at the MS.

1. Create/select a Data Collector Set within PerfMon for the connected AS, which consist of a primary set (e.g. the CPU and Process related performance counters) and a secondary set (e.g. the System, Network, Memory, and Disk related counters). [Tip: give AS a static IP]
2. Ensure that the following applications are installed on MS: PsTools (for executing commands on remote system), WattsUpUSB (for collecting WUP data and modifying settings), the custom scripts e.g. for initiating a measurement run (see Appendix B for an overview).
3. Ensure that MS does not terminate the script prematurely (e.g. execute a test run).

Executing a series of application runs

When both the AS and MS are prepared, a measurement series can be performed, which entails the execution of multiple application runs with a constant rest period between each runs. We have automated the main parts of the related protocol by means of a script, named *PiBatch*, including: (1) initiating and terminating the monitoring with PerfMon, (2) optional rebooting of the system, (3) a parameterized number of Systester execution runs, and (4) the amount of rest between each runs (including a specific amount for before and after the series). Consequently, the possibility of human interference during a measurement has been eliminated. In addition, we investigated the effect of rebooting the system before a measurement series. In a small experiment we found that a system was significantly more active during the first 15 minutes after rebooting. After this period, the system was (in general) stabilized and ready for a measurement. Furthermore, we noticed that by waiting longer, for instance more than 30 minutes, some systems started with processing significantly heavy background processes, which were difficult to determine and, thus, to eliminate. Therefore, we decided to include a minimal of 15 minutes of rest time within the *PiBatch* script before the first application run is initiated. However, at the time of the experimentation, remotely rebooting the *SRVR* system appeared not to be straightforward. Therefore, this activity was not included in the main protocol for executing a series of application runs. This protocol is as follows:

1. When the preparation of the AS and MS is ensured, clear the memory of the WUP meter.
2. Initiate the *PiBatch* script from MS and set the configuration parameters, including: the amount of rest time before/after series and in between runs, the number of runs and Pi decimals, algorithm (Gauss-Legendre or Borwein) and the (hexadecimal) CPU affinity value.
3. When the script is finished, it terminates and saves the logging of PerfMon automatically. Depending on its settings, the WUP continues to collect data until its memory is full.

In order to get a representative dataset for this research, preliminary measurements were performed with each *AppSys*-configurations. Within these measurements, some runs showed considerable differences in comparison to other runs within their series. Therefore, the decision was made to perform measurements until at least thirty clean runs per configuration were collected.

Post-processing the measurement data

After each series of application runs, the following actions are necessary before the recorded measurement data can be used for analysis.

Read out WUP and convert PerfMon logs After the *PiBatch* script is finished, the WUP data is collected via WattsUpUSB and the PerfMon logs are converted to CSV format.

Determine run intervals We use the %CPU Time of the application process, which is recored by PerfMon, to determine the execution times of the application. The execution interval starts at the second at which the %CPU Time of the application process is more than 0 and finishes at the second at which the %CPU Time of the application process is (near) 0 again.

Synchronize WUP and PerfMon timestamps Despite the different options of the WUP for assigning timestamps, WUP data and the PerfMon data do not fit easily. As a solution, we search for the first initiation of a run that has a clear and immediate increase in the WUP data log and adjust its time to the value at the moment of initiation of the related process (in the PerfMon data log). The time in PerfMon is leading as this log includes more details per second: we do not know for sure what causes an increase in the data of the WUP. However, when we have synchronized the starting point of one run, we can also verify if the end points correspond, and repeat the process at the subsequent runs in the series.

Assess quality of runs Despite our efforts to stabilize the systems, we observed that systems can have remarkably variable power usages during the measurements that are often unrelated to S^A. Figure 6.2 shows an example of the energy consumption during a series containing 12 application runs. In the first 15 minutes of the series the variability is caused by the system reboot. In the next 45 minutes, the system executes 4 runs with an average $P(\mathrm{S}^A)$ below 19 W. These runs are considered to be clean as no other unrelated processes were active, and the application base energy use around these runs is similar to the values observed at other measurements with this particular system. The subsequent 7 application runs are considered not to be clean as the increased base energy values around these runs differs significantly.

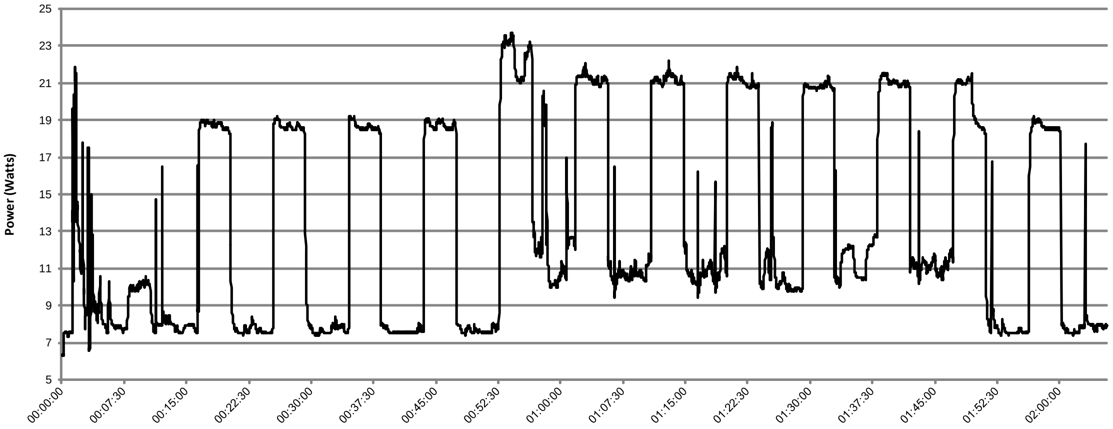


Figure 6.2: WUP data - Series of measurement runs

6.2 Experimentation Results

Executing the protocols, as described in the previous section, resulted in an extensive dataset. Table 6.3 presents the main outcomes of the measurement data, which includes the following elements: the total and “clean” number of runs per application per system, the amounts of time and energy the systems require to execute the applications on average (including median, min and max of the energy values), and additional information with respect to the stability of measurements (by means of their standard deviation). This includes the variability in execution time and required energy between runs, and the average stability of power use during runs. Within the table, the results are grouped per system and each of these groups consists of three rows i.e. one for each application version. Then, the columns are divided over 4 categories, namely: the number of measurements runs per combination, the execution time, the energy use, and a single column for the power use during a run. Subsequently, these categories will be discussed in a similar order.

First, the results show that the number of accepted, or “clean”, runs does not always equal the total amount of runs executed, since sometimes the amount of unrelated workload and/or power use during a run was unacceptable. However, we ensured that at least 30 clean runs per *AppSys* were collected. Because system *SRVR* produced a considerable smaller number of clean runs compared to the other systems, we had to perform additional series of runs with *SRVR*. Remarkably, this problem did not occur with application version *Borw8M2C*. Thus, the additional energy use problem with system *SRVR* only occurs when a single CPU is being used for S^A. Second, the results in Table 6.3 show a general trend with regards to the required execution time and total energy: the *Gauss-Legendre* algorithm is the most efficient of the 3 application versions. Next comes the dual-core version of the *Borwein* algorithm, which is more efficient (both in execution time and total energy) than the single-core version. To illustrate these correlations, we included Figure 6.3 and Figure 6.4, which depict the average amount of time and energy, respectively, a system requires to execute a certain application version (incl. min. & max. bars).

Table 6.3: Overview of the main measurement results per application per system

S = *LPTP*	# Runs		Execution Time		$E(\mathrm{S}^A)$ :: Energy Use (Joules)						$P(\mathrm{S}^A)$ :: Power Use (W)	
Configuration	Total	Clean	Seconds	$\sigma(T)$	Mean	Median	Min.	Max.	$\sigma(E)$	σ/Mean	Mean	Mean(σ p/run)
Gaus8M1C	32	32	137,3	0,46	7.442	7.445	7.399	7.496	20,53	0,0072	54,2	1,53
Borw8M1C	32	32	324,9	0,70	17.989	17.994	17.912	18.096	46,69	0,0047	55,4	1,44
Borw8M2C	32	32	247,8	0,71	14.724	14.712	14.586	14.836	50,53	0,0067	59,4	5,05

S = *DKTP*	# Runs		Execution Time		$E(\mathrm{S}^A)$ :: Energy Use (Joules)						$P(\mathrm{S}^A)$ :: Power Use (W)	
Configuration	Total	Clean	Seconds	$\sigma(T)$	Mean	Median	Min.	Max.	$\sigma(E)$	σ/Mean	Mean	Mean(σ p/run)
Gaus8M1C	34	31	134,0	0,31	10.891	10.906	10.704	11.026	78,56	0,0028	81,3	1,36
Borw8M1C	32	31	317,9	0,34	26.092	26.080	25.876	26.314	121,86	0,0026	82,1	1,27
Borw8M2C	32	31	241,8	0,51	20.555	20.596	20.300	20.808	137,31	0,0034	85,0	4,25

S = *SRVR*	# Runs		Execution Time		$E(\mathrm{S}^A)$ :: Energy Use (Joules)						$P(\mathrm{S}^A)$ :: Power Use (W)	
Configuration	Total	Clean	Seconds	$\sigma(T)$	Mean	Median	Min.	Max.	$\sigma(E)$	σ/Mean	Mean	Mean(σ p/run)
Gaus8M1C	80	42	164,1	1,68	44.870	44.828	43.458	46.153	459,55	0,0102	273,4	1,05
Borw8M1C	72	42	376,2	2,25	103.165	103.252	101.236	104.685	743,21	0,0072	274,3	0,98
Borw8M2C	55	55	295,4	1,37	82.204	82.183	81.269	83.232	402,41	0,0049	278,2	3,78

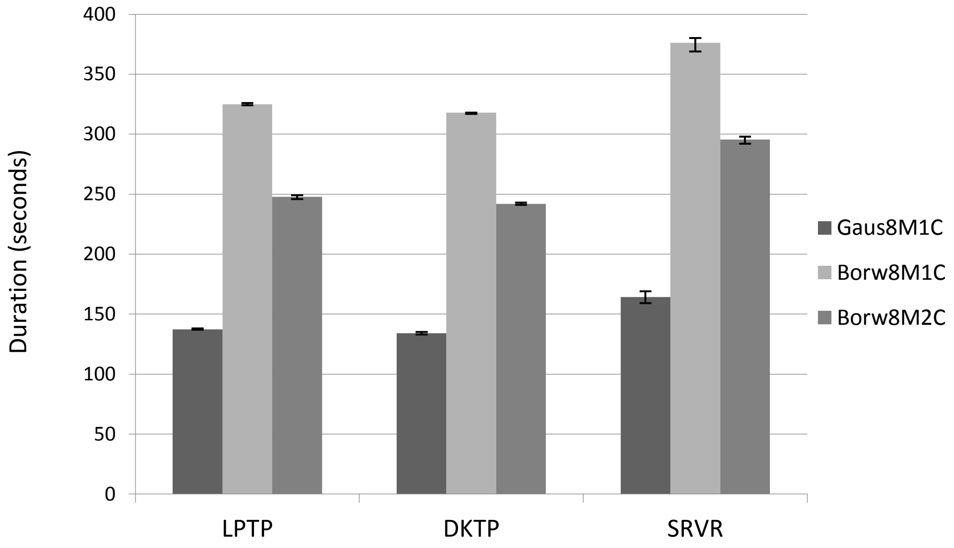


Figure 6.3: Results - Execution time per application per system

Inspecting the stability of the measurement runs, Table 6.3 indicates that system *SRVR* is less stable regarding the required execution time and energy use values. However, when it comes to the stability of the power during single runs (see last column), *SRVR* appears to be slightly more stable than the other systems. In addition, we observed that there are relatively large differences between the required energy use values of the different systems due to differences in scales. Consequently, we calculated the normalized standard deviation values, i.e. σ/Mean, to be able to compare these values on a (more) similar scale. The normalized standard deviation is calculated by dividing the St.Dev.($E(\mathrm{S}^A)$) by the related the $E(\mathrm{S}^A)$ value. This measure is also known as the *unitized risk* or the *coefficient of variation* [6]. The σ/Mean values show that (1) the standard deviation values are insignificant when compared to the absolute means and (2) the normalized deviations between systems appear to be quite similar despite that this is not the case at the absolute standard deviations.

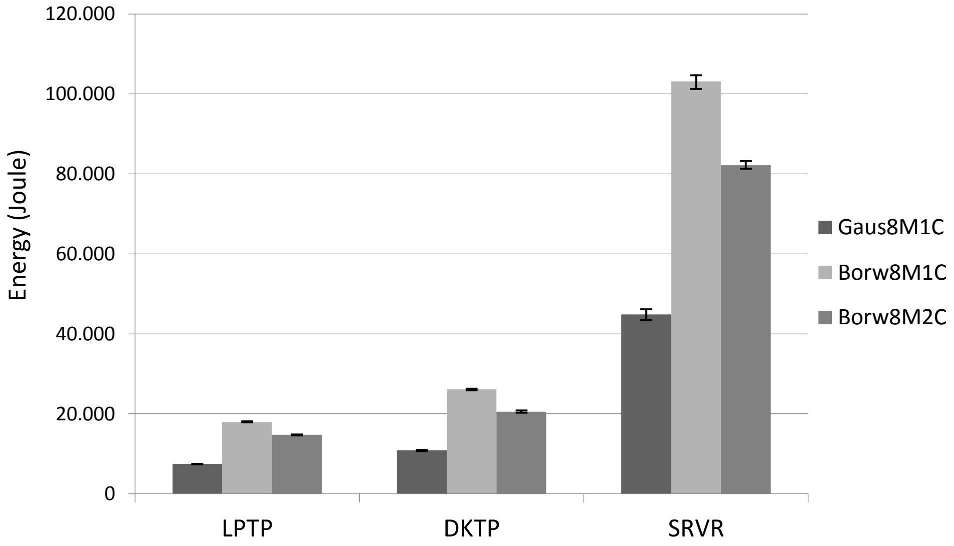


Figure 6.4: Results - Energy use per application per system

6.2.1 Measuring the Base Energy Use Values

Table 6.4 shows the aggregated measurement results related to clean intervals in which systems were in their base state. Because of the variable nature of the systems with regards to their power use, we determined a specific $P(\mathrm{S}^B)$ value during each series of measurement runs. This way, we envisioned to use a more representative $P(\mathrm{S}^B)$ value within each series of an *AppSys*-configuration. However, afterwards we concluded that the differences between the minimum and maximum $P(\mathrm{S}^B)$ values were 0,5, 0,9 and 0,1 for the *LPTP*, *DKTP* and *SRVR* system, respectively. Thus, based on these findings, we decided to use a single $P(\mathrm{S}^B)$ value per system, which simplified the calculations.

Table 6.4: Overview of power during base state

System	$\Sigma\ T(\mathrm{S}^B)$	$P(\mathrm{S}^B)$	Min.	Max.	$\sigma(P(\mathrm{S}^B))$	σ(P) p/run
LPTP	05:36:14	**34,4**	34,04	34,99	0,23	0,20
DKTP	06:28:00	**69,3**	68,09	70,51	0,58	0,92
SRVR	16:13:44	**265,6**	264,99	266,97	0,32	0,35

6.2.2 Determining the Power Use Range

As explained in Section 5.2.2, we would like to normalize the power use values further to minimize the differences in scales. To achieve this, we needed to determine the power use range of a system, i.e. $P(\mathrm{S}^R)$, which lays between the $P(\mathrm{S}^B)$ and $P(\mathrm{S}^M)$, i.e. the maximum power use of a system.

Table 6.5: Overview of determined $P(\mathrm{S}^M)$ and $P(\mathrm{S}^R)$ values (in Watts)

System	$P(\mathrm{S}^M)$	Min.	Max.	σ(P)	ΣT (sec)	$P(\mathrm{S}^R)$
LPTP	**71,5**	69,9	71,7	0,803	883	**37,2**
DKTP	**92,2**	91,4	93,4	1,307	501	**23,6**
SRVR	**288,1**	283,2	291,2	0,801	1936	**22,5**

Table 6.5 shows the maximum power use values which were measured during maximum workload intervals produced by HeavyLoad [1]. Compared to the $P(\mathrm{S}^B)$ and $P(\mathrm{S}^A)$ values, it was more difficult to determine a constant value for the $P(\mathrm{S}^M)$ per system, because the temperature and efficiency of the system are more variable at maximum workload. Consequently, we performed a series of runs to determine the average value for each system. For each system we calculated the difference between the $P(\mathrm{S}^B)$ and $P(\mathrm{S}^M)$ values, denoted $P(\mathrm{S}^R)$, i.e. a system's *power use range.*

6.3 Experimentation Analysis

Next step is to relate the obtained results with the metrics defined in Section 5.2.2. This includes (1) the determination of $E(\mathrm{A}_S)$, (2) the application of the power use range of each system for better comparison of power use values between different systems, and (3) how this can be used to determine the relation between the workload at a system during S^A and the resulting power use.

6.3.1 Determining the Operational Energy Use

As expected, Table 6.3 showed that executing the applications on different systems result in various power consumption and execution time values, despite that the CPUs operated on the same frequencies. Nevertheless, per *AppSys*-configuration, the measurement data is remarkably stable

[1] http://www.jam-software.com/heavyload/

in terms of the required execution time and energy per run, as well as the power during the runs. Furthermore, the results show that the systems require considerable different amounts of energy for executing the application. This can be explained i.a. by the different base values of systems, as is shown in Table 6.4. In response to this, we introduced a more independent metric in Section 5.2.2. The main assumption to come to metric (5.3) is that the system does not execute any other, unrelated, processes while executing the application. Comparing the processing times for the application during runtime, depicted in Table 6.6, shows that, for the single core versions, one core is completely occupied by the process and with *Borw8M2C* it is more evenly balanced. However, during each S^A-measurement, both cores often have a higher workload than during the S^B-measurements. Hence, it was necessary to investigate which processes were active during S^A.

Table 6.6 shows that, during S^A, each system spent less than 4,5% of their %CPU time on other processes than *Systester*, i.e. *App.*, and the *Idle* processes. It also shows that, during the Base intervals (e.g. last row), each system spent less at least 99.7% of their %CPU Time on the *Idle* process. Further analysis of the active processes during the execution of the application showed that the *Systester*, *Idle* and general system processes (e.g. the base) together accounted for at least 99,6% of the %CPU Time. Consequently, at most 0,4% of the %CPU time was related to other processes. Therefore, we conclude that the energy increase during S^A is caused by the execution of the application. As the assumption for metric 5.3 holds, we can calculate the operational energy use for each combination of application and system, as shown in Table 6.7. The results show that, even though we filter out the $P(\mathrm{S}^B)$, this metric remains to be dependent on the system, i.e. there are still differences between the systems per applications version when comparing the $E(\mathrm{A}_S)$ values.

To illustrate the differences, Figure 6.5 gives a graphical comparison between the $E(\mathrm{S}^A)$ and $E(\mathrm{A}_S)$ values. Given that the scales of the 2 y-axis are considerably different (e.g. 120.000 vs. 8.000 as maximum), we argue that the differences between $E(\mathrm{A}_S)$ values are much smaller between systems and applications in comparison to the $E(\mathrm{S}^A)$ values. For instance, the normalized standard deviation of the $E(\mathrm{A}_S)$ per application version, denoted "σ/Mean", is on average 0,30, while the deviation of the $E(\mathrm{A}_S)$ value is 0,79. With respect to the $P(\mathrm{A}_S)$ values, showed at the right side of Table 6.7, we observe a similar significant difference when comparing its average normalized standard deviation value with the specific value of the $P(\mathrm{S}^A)$, i.e. 0,34 vs. 0,71 respectively.

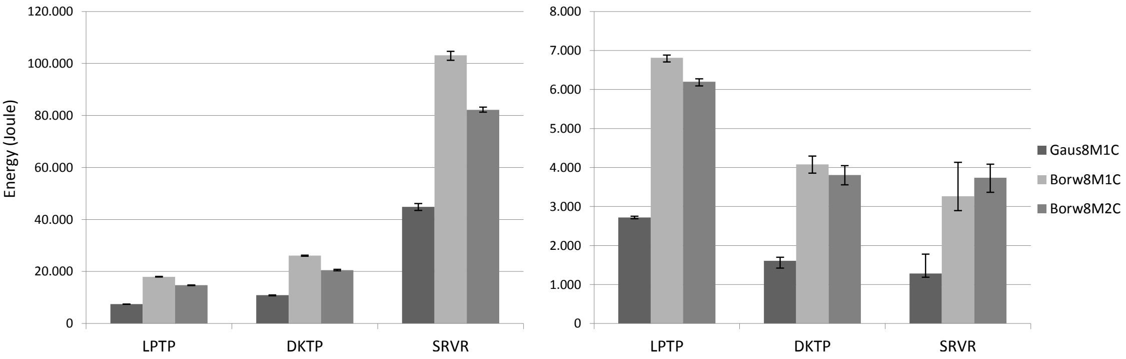


Figure 6.5: Overview of $E(\mathrm{S}^A)$ [left] and $E(\mathrm{A}_S)$ [right] values

Moreover, Table 6.7 shows that, at each application version, *SRVR* and *LPTP* produce the highest and lowest $E(\mathrm{S}^A)$ values, respectively, while at the $E(\mathrm{A}_S)$ the exact opposite is the case. Also, we noticed that at systems *LPTP* and *DKTP* the $E(\mathrm{S}^A)$ and $E(\mathrm{A}_S)$ values related to application *Borw8M2C* are lower, despite that these systems have higher $P(\mathrm{S}^A)$ values during the *Borw8M2C* application version than during *Borw8M1C* (e.g., high but short intensity). However, this does not hold for *SRVR* since *Borw8M2C* produces both the highest energy and power values.

Table 6.6: %CPU Time during measurements per CPU [left] and per Process [right]

	Per CPU						Per Process					
	LPTP		*DKTP*		*SRVR*		*LPTP*		*DKTP*		*SRVR*	
Configuration	cpu_0	cpu_1	cpu_0	cpu_1	cpu_0	cpu_1	*App.*	*Idle*	*App.*	*Idle*	*App.*	*Idle*
Gaus8M1C	4,5%	99,4%	99,0%	3,1%	98,8%	6,3%	49,6%	48,1%	49,4%	49,0%	49,2%	47,4%
Borw8M1C	4,2%	99,7%	99,5%	3,1%	99,4%	6,3%	49,7%	48,1%	49,7%	48,7%	49,5%	47,1%
Borw8M2C	68,0%	69,1%	75,5%	60,2%	59,1%	79,4%	66,2%	31,4%	66,1%	32,1%	64,8%	30,7%
Base	0,2%	0,4%	0,3%	0,3%	0,6%	0,1%	0,0%	99,7%	0,0%	99,8%	0,0%	99,7%

Table 6.7: Overview of determined $E(\mathrm{A}_S)$ [left] and $P(\mathrm{A}_S)$ [right] values

	Energy use :: $E(\mathrm{A}_S)$ (Joules)				$E(\mathrm{S}^A)$	Power use :: $P(\mathrm{A}_S)$ (Joules/s)				$P(\mathrm{S}^A)$
Configuration	*LPTP*	*DKTP*	*SRVR*	σ/Mean	σ/Mean	*LPTP*	*DKTP*	*SRVR*	σ/Mean	σ/Mean
Gaus8M1C	2.719	1.608	1.282	0,3290	0,8017	19,8	12,0	7,8	0,3764	0,7160
Borw8M1C	6.812	4.078	3.261	0,3219	0,7821	21,0	12,8	8,7	0,3608	0,7105
Borw8M2C	6.199	3.806	3.740	0,2496	0,7796	25,0	15,7	12,7	0,2948	0,6934

Table 6.8: Ratios of $P(\mathrm{A}_S)$ to $P(\mathrm{S}^A)$ [left] and $P(\mathrm{S}^R)$ [middle], and %CPU Times [right]

	$P(\mathrm{A}_S)/P(\mathrm{S}^A)$			$P(\mathrm{A}_S)/P(\mathrm{S}^R)$ :: $R(A_S)$			%CPU Time(S^A)		
Configuration	*LPTP*	*DKTP*	*SRVR*	*LPTP*	*DKTP*	*SRVR*	*LPTP*	*DKTP*	*SRVR*
Gaus8M1C	36,5%	14,8%	2,9%	53,2%	50,8%	34,7%	51,9%	51,1%	52,6%
Borw8M1C	37,9%	15,6%	3,2%	56,3%	54,3%	38,5%	51,9%	51,3%	52,9%
Borw8M2C	42,1%	18,5%	4,6%	67,2%	66,6%	56,2%	68,6%	69,3%	69,3%

Further, Table 6.8 (see previous page) includes multiple ratio's. The first one (columns 2 to 4) concerns the ratio between $P(\mathrm{S}^A)$ and $P(\mathrm{A}_S)$ values, i.e. the portion of $P(\mathrm{S}^A)$ related to $P(\mathrm{A}_S)$. By comparing the values per application, we concluded that there is little correlation between systems when the system's total power usage is included, due to large differences in $P(\mathrm{S}^B)$ values between systems. In addition, we see little correlation between these values and their related % CPU Time values (see columns 8 to 10). Consequently, to cope better with different systems, we determined the position of the $P(\mathrm{A}_S)$ value within the specific power use range of systems, i.e. $P(\mathrm{S}^R)$. Hence, column 5 to 7 in Table 6.8 and Figure 6.6 (right) show the resulting $R(A_S)$-ratios.

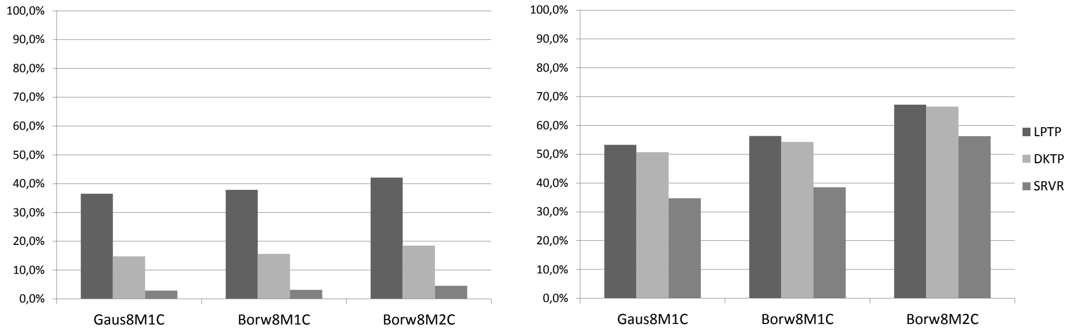


Figure 6.6: Overview of $P(\mathrm{A}_S)/P(\mathrm{S}^A)$ [left] and $R(A_S)$ [right] ratios

The values in the table and graph show that there is a stronger correlation between the systems if we normalize the operational power use values of each application to the respective power use ranges (i.e. metric 5.4). For instance, the difference between the minimal and maximum values per version decreases by 19,5% (e.g. averaged over the different systems). In addition, Figure 6.7 illustrates that there is a stronger correlation between the $R(A_S)$ and %CPU Time values (per system). For instance, while the min/max-difference between $P(\mathrm{S}^A)/P(\mathrm{A}_S)$ and %CPU Time is 50,7% on average with a standard deviation of 16%, the difference between $R(A_S)$ and %CPU Time is only 17,6% with a deviation of 6%. Furthermore, statistics (i.e. *Pearson's r*) show a significant correlation with *Gaus8M1C* at *LPTP* ($c : 0.367, p : 0,043$) and at *DKTP*($c : -0.366, p : 0,04$).

These results indicate that metric 5.4 can help to align CPU workload with the resulting power use. However, the results also indicate that the metric is not yet stable enough to be used as an system independent metric for sustainability. Analysis using one-way *ANOVA* on the $R(A_S)$ values only shows no significant difference between the *Gaus8M1C* values of *LPTP* and *DKTP* ($p : 0,07$). A possible cause for the lack of more similarity can be that our assumption of a linear dependence between the workload and the energy usage does not hold, or that the measuring of the maximum power consumption did not correctly take all the hardware components into consideration.

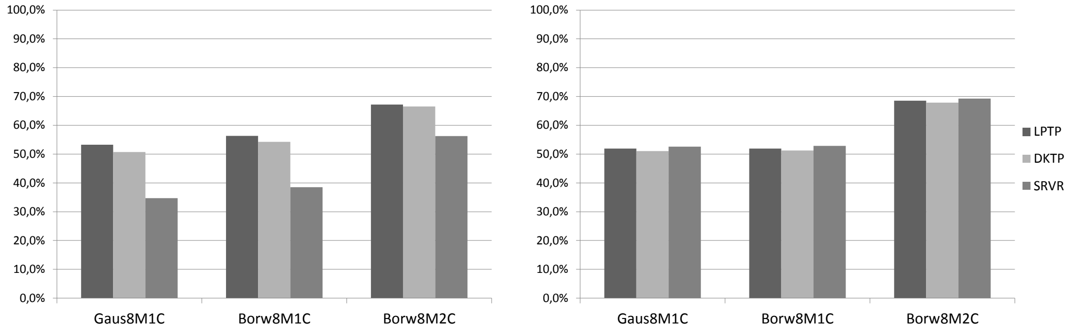


Figure 6.7: Overview of $R(A_S)$ [left] and %CPU Time(S^A) [right] ratios

Table 6.9: Differences between applications :: $T(\mathrm{S}^A)$ [left] and $E(\mathrm{A}_S)$ [right]

	Execution time :: $T(\mathrm{S}^A)$			Energy use :: $E(\mathrm{A}_S)$		
Ratio $f(A_x)/f(A_y)$	*LPTP*	*DKTP*	*SRVR*	*LPTP*	*DKTP*	*SRVR*
Gaus8M1C/Borw8M1C	42,3%	42,2%	43,6%	41,0%	39,4%	39,3%
Gaus8M1C/Borw8M2C	55,4%	55,4%	55,6%	44,9%	42,3%	34,3%
Borw8M2C/Borw8M1C	76,3%	76,1%	78,5%	91,5%	93,3%	114,7%

6.3.2 Differences between Application Versions

Besides the system independence of the metrics, we also require to investigate their rigidity. To do so, we determined the ratios based on the differences in required execution time and energy between different application versions. This enables us to determine whether differences between application versions remain the same between different systems. Table 6.9 shows that the differences in required time for executing the application versions are relatively similar, which is also depicted in Figure 6.8. However, *SRVR* often deviates (slightly) from the others, which could be explained by lower RAM performance compared to the other systems. The same holds for the $E(\mathrm{A}_S)$ values of systems. Here, we only observe a strong similarity between the three systems at the *Gaus8M1C/Borw8M1C*-ratio. Further, ratios that include *Borw8M2C* show weaker correlations between the systems, mostly due to *SRVR*. Therefore, we assume that *SRVR* has less advantage of dual-core processing, since only at *SRVR* holds that E_{A_S}*(Borw8M1C)* < E_{A_S}*(Borw8M2C)*.

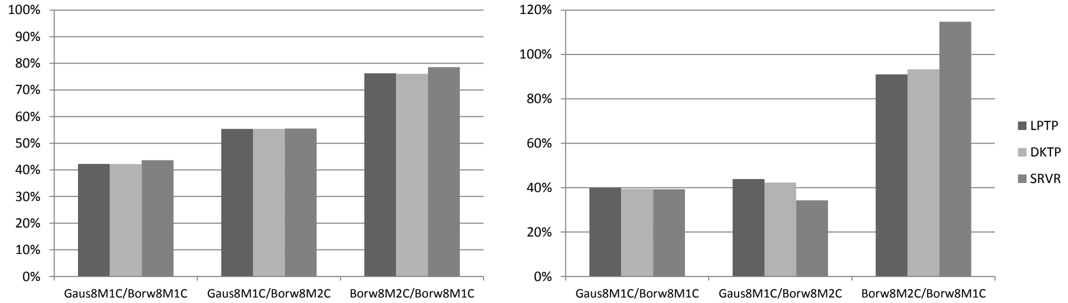


Figure 6.8: Differences between applications :: $T(\mathrm{S}^A)$ [left] and $E(\mathrm{A}_S)$ [right]

6.3.3 Investigating the Other Components

As mentioned in Section 6.1, we also recorded the performance of other components during S^A. Although the exact relation between (outstanding) values at these performance counters and the (resulting) $E(\mathrm{S}^A)$, let alone $R(A_S)$, values is beyond the scope of this research, we discuss several elements concerning these additional recordings, including: (1) an example of a non-clean application run which showed remarkable values at other performance counters than the %CPU Time, and (2) differences at other components between different systems and versions of *Systester*.

Figure 6.9 shows the energy consumption of system *DKTP* related to a series with 16 executions of application version *Borw8M1C*. In this series, the first run was considered not clean and the remaining 15 runs were judged as clean runs. Regarding execution time and required energy, there was already a noticeable difference between the first run and the other runs, namely: 403 seconds instead of 318 seconds and 34.633 Joules instead of the 26.141 Joules on average at the remainder of the runs. Table 6.10 shows a selection of non-CPU Time performance counters, that contained unusual values during Run 1 im comparison to the other runs in this particular series.

Table 6.10: Overview of (average) non-CPU performance values at *DKTP*

	System (per second)			Disk	Memory	NIC
	System calls	Data operations	Contex switches	%Idle	Pages/s	Bytes/s
Run 1	13.530	126,16	2.690	43,2%	278	30.671
Rest	2.339	9,13	1.054	100%	0,05	29.724
σ(Rest)	82,58	0,51	12,39	0,02	0,10	170,05
Base	2.347	9,23	940	100%	0,37	29.597

For instance, we see additional activity at the Disk, Memory, NIC, and at the three System performance counters (Nb. in fact, almost all of the measured System counters showed unusual values during Run 1). In addition, the σ *(Rest)*-row shows, for each particular performance counter, the amount of deviation between the values of the "other" runs. Due to the relatively low values in this row, we can conclude that the specific values in the *Rest*-row give a reliable indication of the average values. We assume that the Disk component is the main cause for the increased energy values, since it consumes a substantial amount of power when it is active [33]. However, given the available instrumentation, we are currently not able to further validate this assumption.

Moreover, we assume that the analysis of the performance counters of non-CPU components can serve 2 main purposes: (1) investigating the cause(s) for an increased energy use if the %CPU Time does not show unusual values and/or (2) investigating the possible source(s), e.g. processes, in case the %CPU Time does show unusual values. The first example in this section belongs to the latter purpose. It shows that, even with all these additional counters, it remains (too) difficult to determine the exact cause(s). Nevertheless, we argue that by using PerfMon in combination with the WUP meter we can sufficiently determine whether a run is clean, i.e. by assuring that the activity at the CPU is caused by the specific application and that the other components do not show "abnormal" activity, which entails activity that is not similar to the *workload profile* of the application. For instance, the profile of the single-core *Systester* is that it utilizes one (of the) CPU(s) with a maximum intensity, while the other components should remain relatively idle.

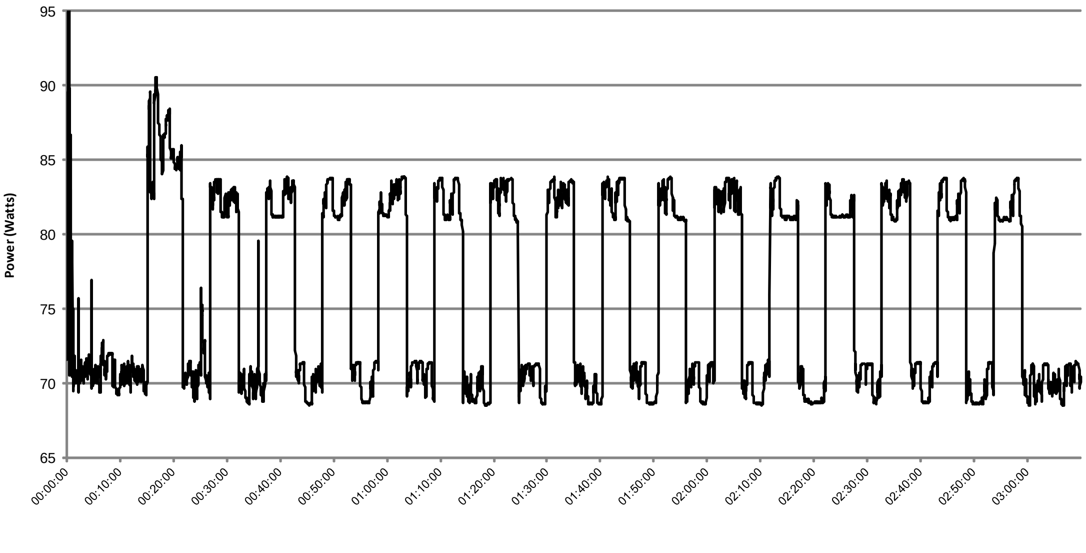


Figure 6.9: WUP data - 16 runs of application *Borw8M1C* on system *DKTP*

Table 6.11: Overview of average non-CPU performance values at *LPTP* and *DKTP*

	#	System (per second)			Disk	Memory	NIC
App/Base	R.	System calls	Data operations	Contex switches	%Idle	Pages/s	Bytes/s
A(*LPTP*)	32	683,9	10,2	331,4	100%	0,05	30.671
A(*SRVR*)	33	954,1	11,1	222,1	100%	0,75	28.963
B(*LPTP*)	18	559,8	10,5	404,1	100%	0,08	33.903
B(*SRVR*)	19	845,7	9,07	197,6	100%	0,07	28.565

To investigate the workload profile of *Systester* further, Table 6.11 shows the non-CPU performance counters values related to a series of measurements performed with *LPTP* and *SRVR*. Similar to the previous table, we combined the values related to application runs to compare them with the values related to the base. The values indicate that the System and NIC related values vary between the different systems with respect to the relationship between application and base runs (e.g. which value is greater). Therefore, we cannot make any hard conclusions about these counters. However, we do see that the Disk and Memory components remain relatively idle during both the (clean) application and base runs. Since, the same holds for the values at *LPTP*, we are confident that these specific characteristics should be included in the workload profile for *Systester*.

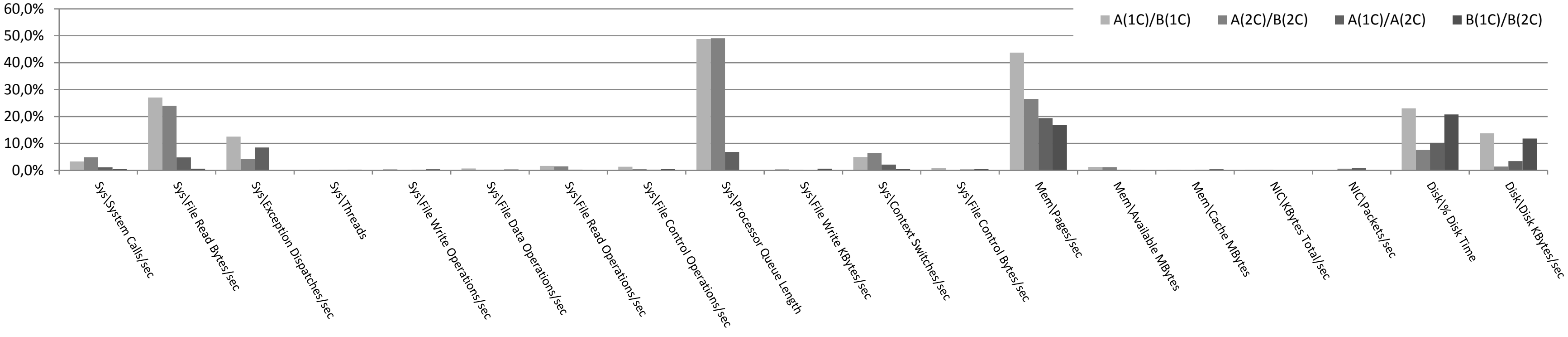
Finally, we compare the non-CPU performance counter values of the Borwein single/dual-core versions of *Systester*. For this, we selected two series of measurements performed by *LPTP*, which both contain 32 clean application runs. Table 6.12 shows the values of all the non-CPU performance counters, which were included during the experimentation (see also Appendix A). Since the scale between the counters is different, we normalized the values to be able to compare the amount of difference (i.e. $abs(single - dual)/\frac{single+dual}{2}$) between the counters. Consequently, we concluded that the largest relative differences between the two versions are at Processor Queue Length (0,62), Disk Time (0,49), Exception Dispatches/s (0,31), and Memory Pages/s (0,30). However, it should be noted that due to the large differences in scale, this comparison method is not ideal, because the absolute values closer to zero result in a higher relative difference more quickly.

In addition to this table, the next page also includes a graph, which illustrates the relative differences between the counters of the versions. Furthermore, to handle the differences in scales between counters, we applied the same formula from the previous paragraph, which divides the absolute difference between two values by their mean. Besides the difference between the values related to the application runs of both versions, i.e. *A(1C)/A(2C)*, we also included the following comparisons: singe-core application vs. (single) base (*A(1C)/B(1C)*), dual-core application vs. base (*A(2C)/B(2C)*), and (to be complete) single-core base vs dual-core base (*B(1C)/B(2C)*). This graph quickly shows, or underlines, that the largest relative differences are at the Processor Queue Length, Memory Pages/s, Disk Time, Exception Dispatches/s, and File Read Bytes/s counters. Furthermore, it shows that at most counters the differences between the application runs and base runs is larger than the difference between the two application versions. However, at some counters the opposite is the case and therefore it remains difficult to draw exact conclusions from such a graph. Nevertheless, with respect to the provided table and the graph, we conclude this section with the following interpretation: since two CPUs are used to process *Systester*, the dual-core version has a smaller Processor Queue Length value (e.g. less waiting time), higher Memory Page/sec and %Disk Time value (e.g. higher utilization and quicker results). Consequently, besides the CPU component, the Disk and Memory components are more active during the dual-core version and, therefore, the systems requires (on average) more power during the execution. However, due to a better utilization of the CPU and Memory components, most systems require less time and, thus, less energy (e.g. both $E(\mathrm{S}^A)$ and $E(\mathrm{A}_S)$) to execute the *Borw8M2C* version.

Table 6.12: Overview of workload at other components during measurements with *Systester* and *LPTP*

Comp.	SYSTEM												MEMORY			NIC		DISK	
Config. (*Borw.*)	System Calls/sec	File Read Bytes/sec	Except. Dispatch/s	# Threads	File Write Operations	File Data Operations	File Read Operations	File Control Oper.	Processor Queue L.	File Write KBytes/s	Context Switches/s	File Control Bytes/s	Pages/sec	Available MBytes	Cache MBytes	KBytes Rec./sec	Packets Sent/sec	% Disk Time	Disk KBytes/sec.
**8M1C*	682,9	510,6	0,00	488,1	8,13	10,2	2,04	64,8	1,60	30,3	331,4	3,93	0,05	1.590	19,4	34,2	27,4	0,17	2,35
Base	599,8	1.712	0,01	485,6	8,28	10,5	2,18	68,4	0,02	30,9	404,1	4,08	0,75	1.674	19,5	33,9	27,5	0,46	4,14
**8M2C*	716,0	619,6	0,00	483,9	8,18	10,2	2,06	65,5	2,10	30,4	304,5	3,99	0,11	1.603	19,2	34,3	28,3	0,26	2,71
Base	589,2	1.756	0,01	480,6	8,15	10,3	2,19	66,9	0,02	30,2	394,9	4.011	0,37	1.683	19,2	33,9	27,6	0,19	2,56

Figure 6.10: Relative differences between versions (normalized to the mean of the specific combination) with *LPTP*



6.4 Additional considerations

During the course of the experimentation phase, we have considered many additional aspects that have not been discussed yet. This includes an attempt on determining the power consumptions at different workload levels (e.g. to gain a more detailed profile) and also how an artificially increased base workload (e.g. +50%) would influence the execution of an application. To provide additional information to the "enthusiastic" readers, we included a select few of these aspects in this section.

6.4.1 Power Profiling the Systems

As we mentioned before, we assumed a linear dependence between the system workload (at the CPU component) and the resulting power consumption to determine the $R(A_S)$ values. One of the reasons for this, is that it remains difficult to determine the power consumption related to an exact workload at a system or a specific component. We have showed that the power consumption during base workload can be achieved, but still provides obstacles, and that the maximum workload is more difficult mostly due to the influence of variable temperatures. Further, the workloads between the base and maximum workload are even harder to determine, because most stress test tools such as *HeavyLoad* are designed to create only a maximum workload at a set of components. In addition, given that there are multiple components, there are many different combinations to consider (e.g. 50% CPU, 0% Disk, ... vs. 50% CPU, 25% Disk, ... vs. 50% CPU, 50% Disk, ...).

After an extensive search, we finally found *Pressure* [2], which is a CPU stress tester that enables the artificial creation of workload per separate CPU core and, thus, specific %CPU Time values. In addition, since it can be installed as a Windows service and initiated from command-line, we were able to remotely conduct measurements with it. Given our focus on the CPU component and the additional functionalities, we happily settled with this tool. However, due to the OS of *SRVR*, we were not able to install the service on this system and, therefore, it is not included within the results. Further, since *Pressure* only controls the CPU component, we realize that the power use related to its maximum achievable workload is lower than when *HeavyLoad* creates a maximum workload at all components together. Hence, we should note that the results presented below remain experimental (e.g. until all of the components are included properly) and that they are more valuable when combined with an application that mostly creates workload at the CPU component.

As Figures 6.11 and 6.12 show, *Pressure* enables us to produce various workload steps resulting in related power consumption steps. For instance, the graphs in these figures contains the following steps: 25%, 50%, 75% and 100% workload, at 1, 2, and 4 CPU cores. (Nb. the combination of 1 CPU core at 25% and the other cores at 0%, i.e. the first step, is missing due to an unclean run).

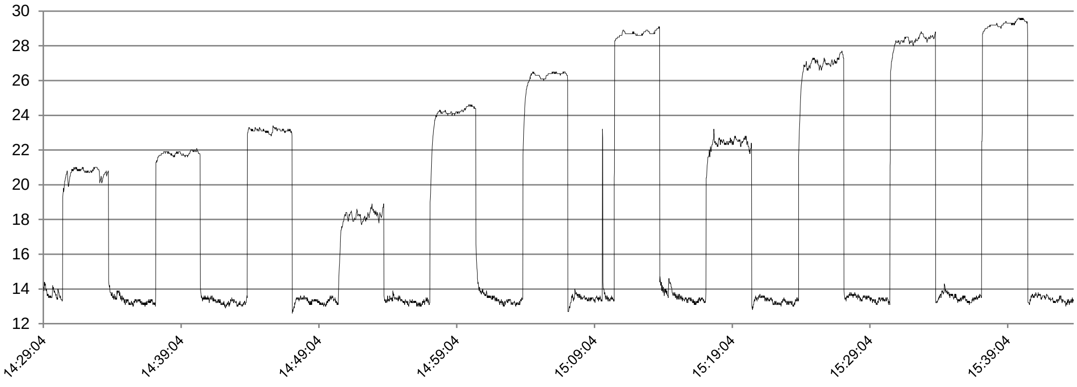


Figure 6.11: WUP data - Series of measurement runs with *Pressure* on *LPTP2*

[2] http://www.elifulkerson.com/projects/pressure-cpu-stress-tester.php

We used these steps to determine the relation between a systems workload and the resulting power consumption at various intensities. In addition, we were able to conduct multiple (automated) measurements, since we already had useful protocols and multiple test systems available. Since *Pressure* did not work on *SRVR*, the set of systems on which we performed the tests included the *DKTP* system and three laptops: *LPTP*, i.e. *LPTP1*, and the two previously excluded laptops with Hyper-Threading technology, i.e. *LPTP2* and *LPTP3*. Furthermore, we only included 4 different steps, due to the differences in cores between the systems and also a limited amount of time. These steps were 2 separate cores at 25%, 50%, 75% and 100%. As can be seen in Figure 6.11, we also investigated the options of using 4 cores (with the two modern laptops), however we quickly realized that since the 4 cores of these systems are "logical", or *virtual*, their behavior is not at all similar to a system with 4 separate physical cores and, therefore, we ignored these options.

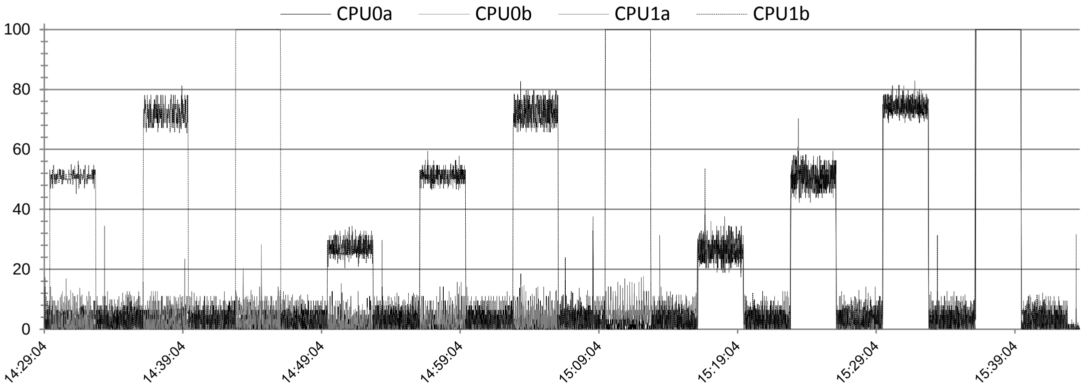


Figure 6.12: PerfMon data - CPU Time per logical cores of *LPTP2*

The results of the measurements are included in Table 6.13 and Figure 6.13, which illustrates two graphs containing absolute and relative values per systems. The absolute graph shows the resulting power consumption at each workload step, while the relative graph shows the normalized values by rescaling the absolute values by the specific range that lies between the minimum and maximum power values per system. Since the absolute values are on different scale, the latter graph is useful because it enables us to compare the values on a single scale between 0 and 1. The results show that the dependence between the workload and power use variables are more closer to a linear growth than to, for instance, an exponential (e.g. what most of us expected) or logarithmic growth. We argue that this observation is in favor of our metrics. However, to be more sure, we should conduct additional measurements with more test systems, more workload steps, and, to further minimize the amount of noise, more and longer iterations (while monitoring the temperature of the systems). Nevertheless, we believe that our (experimental) attempt with *Pressure* can be used as a good foundation for more advanced profiling methods.

Table 6.13: Overview of Pressure results per workload step in absolute and relative values

	Absolute (W)					Relative (%)				
System	0%	25%	50%	75%	100%	0%	25%	50%	75%	100%
DKTP	69,0	75,0	77,2	83,5	86,8	0,0	33,7	46,1	81,5	100,0
LPTP1	35,7	39,5	43,2	45,4	49,1	0,0	28,4	56,0	72,4	100,0
LPTP2	13,0	18,7	24,4	26,5	29,0	0,0	35,6	71,3	84,4	100,0
LPTP3	8,4	13,1	16,6	19,4	22,9	0,0	32,5	56,8	76,3	100,0

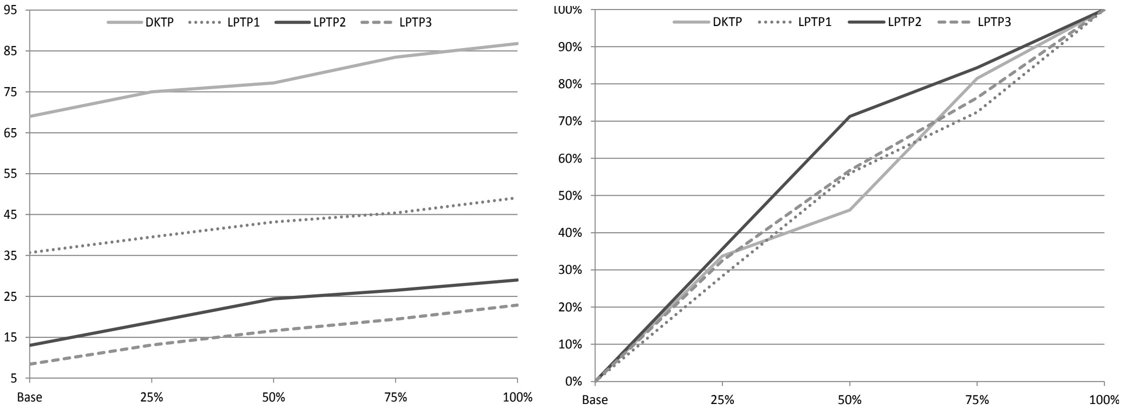


Figure 6.13: Pressure results per workload step in absolute [left] and relative [right] values

6.4.2 Increasing the Base Workload

After our initial measurements with *Pressure*, we also considered performing measurements with *Systester* while in the meantime *Pressure* would produce an artificially increased base workload. The hypothesis behind this investigation would be: if we increase the base workload with 50%, will the execution time and/or $E(\mathrm{A}_S)$ values of *Systester* double in comparison to the original values. The idea behind this hypothesis is related to the dependency between the workload and power consumption variables, because originally we assumed that there would be a more exponential growth between the two variables, which would imply that a system would be less efficient when it reaches its maximum workload and power consumption values. However, Table 6.13 shows that, in fact, the growth has a slight decreasing drop (e.g. it is steeper near the base/minimal workload).

To investigate the hypothesis, we performed additional measurement with *LPTP2*, i.e. a modern laptop with Hyper-Threading technology. We performed 3 different series containing multiple runs during which *Pressure* was configured to produce workloads of 25%, 50% and 100% at two separate physical cores, respectively. Moreover, these specific workloads would also be produced during the base intervals in order to determine the (artificially increased) $P(\mathrm{S}^B)$ values per series.

The results of our investigation showed that the three steps resulted in an execution time of 31%, 80%, and 138% greater than the original value, respectively. Then, with respect to the total required energy, we see that the steps resulted in an increase of 44%, 100%, 271%, respectively. The interesting thing of these values is that although the execution time does not double in value, the required energy in the case of the 50% step is exactly twice the amount of the original value. The reason for is that besides the increase in execution time, the average power consumption also increases when *Pressure* is configured with a higher workload step (despite the fact that *Systester* is designed to consume maximum CPU resources). In hindsight, we realize that *Systester* is not an ideal application for this kind of investigation due to its specific design. Furthermore, we should also note that the Hyper-Threading technology caused for difficulties and therefore it would be better to conduct additional measurements with other systems, that only have physical cores.

6.4.3 Influence of the Operating System

Due to the many alterations of the included systems, we have also gathered measurements data of *DKTP* with a clean 64bit version of Windows 7 installed, besides the 32bit version. Although the set of 64bit-runs is much smaller, we noticed that the execution times are similar to the 32bit version, but that it requires slightly more energy during S^A (e.g. 1,3%). However, since the 32bit has a lower $P(\mathrm{S}^B)$ value, the two versions appear to require similar values solely related to S^A.

Chapter 7

Validation

This chapter describes the case-study conducted after the initial experimentation, i.e. the validation of the metrics. Similar to the activities described in Chapter 6, the case-study includes a set of protocols for measuring and gathering sufficient (and correct) data. However, to test the external validity of the metrics, the case-study was performed with a different experimental setup (e.g. a more complex application). Hence, before we discuss the results and analysis of the case-study measurements in Section 7.2 and 7.3, respectively, we explain the specifics of the setup first.

7.1 Validation Setup

Within this thesis (cf. Section 6.1), an experimental setup comprises the following elements: instrumentation, systems, application and experiment protocols. With respect to the instrumentation, the WUP meter is used again for measuring the power consumptions. As Figure 7.1 shows, the meter is placed between the Application System (AS) and the power source of AS. Furthermore, it is connected with a Management System (AS) to be able to download the energy data and change settings. Further, with respect to the systems and applications, there are some noteworthy differences in comparison to the original experimental setup. First, the only system that plays the role of AS is the *SRVR* system. The reason for this is that the validation application, i.e. *DOCGEN*, is designed for servers (and there was only one server available, unfortunately). Second, we can see in Figure 7.1 that the AS is now connected (at least) with an external Oracle database and with one or more clients. To explain these relations, we briefly discuss the basic workflow of *DOCGEN*.

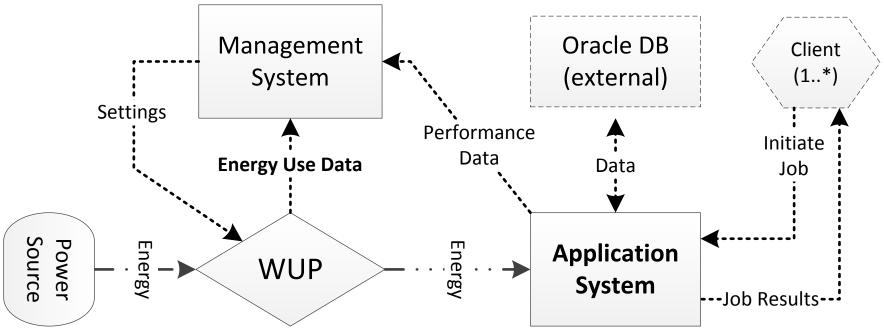


Figure 7.1: Representation of Experimental Setup II

DOCGEN is a mail tool developed by the research partner, that works independently from an ERP system or document management system (DMS). It can be used to automatically generate large batches of documents (e.g., in real life it generates over 30 million documents per year).

When a batch, or *job*, is initiated by the user at the client-side of *DOCGEN*, the server-side (i.e. deployed at the AS) validates the request and, if accepted, starts the collection of the data and documents definitions. This data resides in a separate database, i.e. the Oracle DB in Figure 7.1, whereas the definitions are created and managed internally at the AS. Once collected, the data is merged into a preview, that has to be approved, or the generation of documents is directly started, depending on the configurations of the user. Once the generation is finished, the documents can be archived (e.g. in a DMS) or directly transfered to an outlet (e.g. printer or mail system).

During this case-study, we want to determine the required energy consumption for executing a job in *DOCGEN*. As described above, there are multiple systems involved with the processing of a job. Therefore, it would be best to measure the energy consumption of all these systems. Unfortunately, this was not possible because only one WUP meter was available and, also, we did not have exclusive access to the Oracle DB. However, given our focus on the CPU component, we argue that most of the processing specifically related to *DOCGEN* is performed by the AS. We assume that both the initiation of the job (e.g. at the client) and collection of data (e.g. at the Oracle DB) are relatively constant factors, while the generation of documents (e.g. at AS) might depend heavily on whether the processing is done by a single core or multiple cores. Although we cannot validate this assumption for the Oracle DB, we did attempt to validate it at the client side by monitoring its workload during each job (e.g. via PerfMon). This showed no significant differences.

Finally, with respect to the experiment protocols applied during the validation, some alterations had to made due to the specific experiment environment. First, it was not possible to initiate a job in *DOCGEN* from command-line, let alone with the desired configuration. Second, the job must be initiated from a separate client system (CS), which entails that (if we exclude the Oracle DB) there are at least 3 different systems within the environment. In addition, we considered experimenting with more than one client concurrently. However, due to a limited time for measuring, this remains for future research to be examined. Consequently, the main protocol for *DOCGEN* is as follows:

1. Ensure that *DOCGEN* works correctly at the server-side and clear cached/previous data.
2. Ensure connection between AS and MS (cf. Section 6.1.4) and clear memory of the WUP.
3. Initiate (from the MS system) remote performance logging of both AS and CS.
4. Start the client-side of *DOCGEN*, ensure connection and select a specific job.
5. After waiting a moment to settle from startup (e.g. less interference), initiate the job.
6. When the job is finished (e.g. documents are generated), wait and prepare for a next job.
7. When sufficient jobs have been executed, collect and process all the measurement data.

As protocol steps 6 and 7 describe, we attempted to execute multiple jobs during a measurement series (e.g. similar to the experimentation with *Systester*). However, due to the design of *DOCGEN*, the activity of initiating a next jobs could not be automated. Therefore, we decided to determine the average execution time of a specific job and, then, wait this amount time plus an additional 15 minutes to let the system sufficiently settle before a next job. Furthermore, also due to the design of the application, the preparation of a job required the removal of the results related to a previous job. Otherwise the execution of the next job will be influenced, if it requires to produce the same results as the cached data, which is not acceptable. To minimize interference with the AS, this data is removed remotely through a script initiated from MS (i.e. via *PsTools*).

Further, with respect to a suitable test job, we conducted multiple orientation measurements with jobs that were included in the supplied version of *DOCGEN*. From this, we determined that a specif job containing 5000 documents resulted in a sufficiently long execution time, e.g.: around 40 minutes. We realize that this is much longer than the execution times with *Systester*. However, given the protocol for the measurements with *DOCGEN*, and its expected profiles, we concluded that it would be more practical and valuable to focus on a longer task during the validation.

7.2 Validation Results

Executing the protocol described in the previous section, resulted (again) in an extensive dataset. Only this time, there was one AS, i.e. *SRVR*, and two different application versions, i.e. the single- and dual-core version. To have two different versions of *DOCGEN*, we modified the CPU affinity of the application, since (by design) the application utilizes all available CPUs of the executing system and, unfortunately, it was no option to modify the source code. Therefore, to simulate a single-core version, we applied the same method from the experimentation, which entails configuring the CPU affinity of the main process at AS, i.e. *BriefServer.exe*, to a single CPU.

Table 7.1 presents the main outcomes of the validation measurements. This table includes the same elements as Table 6.3, namely: the total and "clean" number of measurement runs, the amounts of time and energy *SRVR* requires to execute the different applications versions, or *configurations*, on average (including median, min and max of the energy values), and additional information with respect to the stability of the measurements based on their standard deviation.

NB. besides the two expected configurations, the table includes an additional third one, i.e.: *Single-as-Dual*. The reason for this, is that we had difficulties with limiting the CPU affinity of *DOCGEN* to a single CPU. Consequently, the results in the third row relate to measurements that were configured as single-core, but that were eventually executed with dual-core characteristics.

Figure 7.2 depicts a graph which shows the %CPU Time of both CPUs of *SRVR* during a single-core execution, i.e. gray-colored, and a dual-core execution, i.e. black-colored. We can see that the execution can be divided into two parts: the beginning with high intensity and the (relatively longer) ending with low intensity. Furthermore, this graphs shows several differences in characteristics between the two version. The main three of these differences are (1) the beginning part of the single-core version lasts longer and has a higher intensity, (2) the single-core requires more time to execute (e.g 150 seconds), and (3) the intensity at the CPUs of the dual-core version are more balanced during the beginning, which is of course not that surprising (e.g. given the configuration). Finally, (partially) due to the differences in workload at the CPU component, we can also see differences with respect to the energy and power values. For instance, the power use of the single-core version is higher and, in combination with its longer execution time, the average total energy use required for this version is considerably higher. i.e. 682.260 vs. 630.826 Joules. In addition, we see more variability with respect to the execution times in comparison with *Systester*.

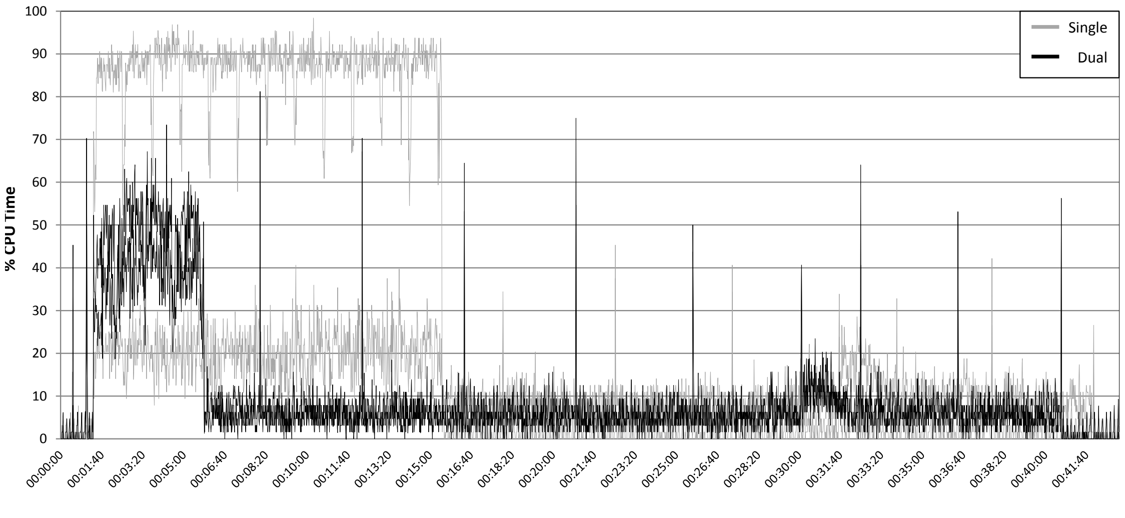


Figure 7.2: Comparing %CPU Time values of different versions during a *DOCGEN* run

Table 7.1: Overview of the main measurement results with *DOCGEN* and *SRVR*

	# Runs		Execution Time		$E(\mathrm{S}^A)$ :: Energy Use (Joules)						$P(\mathrm{S}^A)$ :: Power Use (W)	
Configuration	Total	Clean	Seconds	$\sigma(T)$	Mean	Median	Min.	Max.	$\sigma(E)$	σ/Mean	Mean	Mean(σ p/run)
Dual-core	36	33	2.354	22,71	630.826	630.406	620.070	650.456	6.915	0,0110	268,0	4,09
Single-core	22	19	2.506	90,68	682.260	670.798	656.104	741.732	26.927	0,0395	272,2	5,92
Single-as-Dual	53	47	2.343	22,65	625.731	625.914	615.582	647.498	6.244	0,0100	267,1	3,05

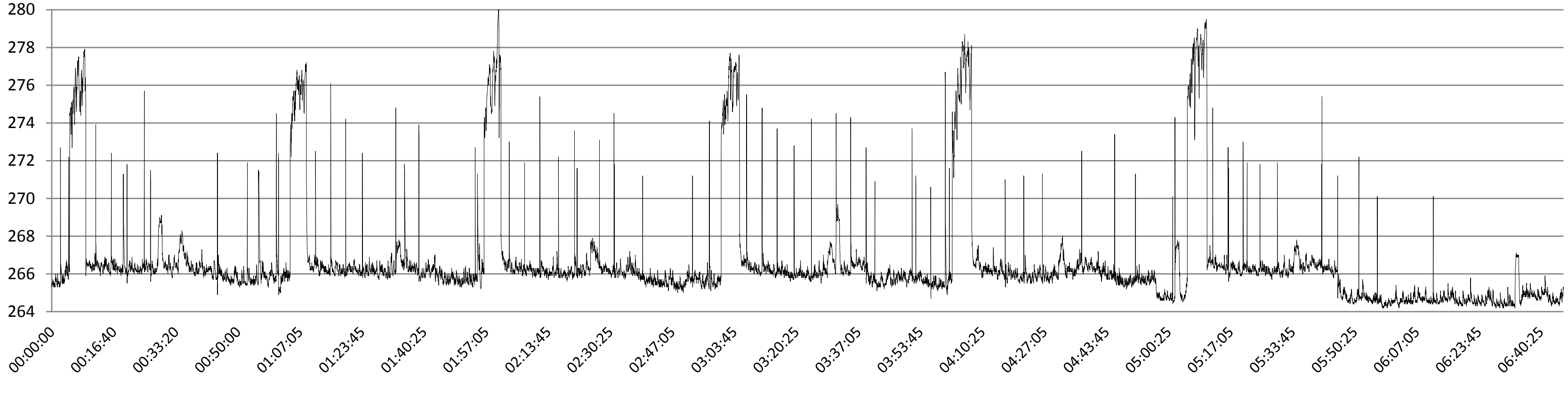


Figure 7.3: WUP data (in Watts) - Series of 6 measurement runs with *DOCGEN*

7.3 Validation Analysis

The analysis within the validation entails testing the metrics defined in Section 5.2.2 with the data resulting from the *DOCGEN* measurements. Since the metrics have already been explained extensively, the analysis has been limited to the following topics: investigating the workload at the CPU component, applying the metrics on the results, and the influence of other components.

7.3.1 Investigating the CPU Component during a Job

Similar to the experimentation with *Systester*, we monitored the performances of the executing system during the validation measurements to be able to tell (more precisely) whether a run was clean. For this monitoring, we used a different data collector, because *DOCGEN* is a more complex application with multiple active processes during S^A. For instance, besides *BriefServer*, *Idle* and *System*, the following processes showed additional activity during S^A: *snmp* (e.g. network management), *lsass* (e.g. local security authentication), and service hosts #2, #6, and #8. The monitoring results, i.e. the average performance values per configuration, are shown in Table 7.2.

Table 7.2: %CPU Time during *DOCGEN* measurements per CPU [left] and per Process [right]

	Per CPU		Per Process							
Config.	cpu_0	cpu_1	*BriefS*	*Idle*	*snmp*	*System*	*lsass*	*svch*2	*svch*6	*svch*8
Dual	7,56%	6,28%	8,99%	86,2%	1,69%	0,28%	1,81%	2,44%	0,23%	0,98%
Single	3,53%	25,4%	24,5%	71,1%	2,03%	0,14%	0,12%	1,40%	0,06%	0,99%
S-as-D	3,70%	7,26%	7,32%	89,0%	1,46%	0,20%	0,06%	2,32%	0,05%	0,94%
Base	0,79%	0,16%	0,03%	99,1%	0,0%	0,09%	0,06%	0,03%	0,05%	0,99%

From Table 7.2, we can conclude that the *Single-as-Dual* configuration is more closely related to the *Dual-core* configuration, than to the *Single*'s. Furthermore, in contrast to our orientational measurements, we see that some of the included processes turned out to be not considerably more active during S^A than during S^B, such as service hosts #6 and #8. However, a reason for this might be that during the Base intervals the connection of *DOCGEN* between the client and server remains active. Nevertheless, based on the performance data, we assume that the influence of these processes is insignificant in comparison to the influence of the *BriefServer* process. Furthermore, we can also conclude that at each configuration the %CPU Time values of the processes sum op to 100%. In fact, they are even a bit more than 100% (e.g caused by rounding errors in PerfMon).

7.3.2 Normalized Operational Energy Use of *DOCGEN*

Now that we have shown that (at least at the CPU component) the workload during the S^A runs is (almost) completely caused by S^A related processes, we can apply the subtractive method for determining the operational energy use of *DOCGEN*. To do so, the $P(\mathrm{S}^B)$ values are required, which we determined for each series separately. However, since the differences between the minimal and maximum measured $P(\mathrm{S}^B)$ values within series were between 0,76 and 1,45 W, we have used a single value per configuration, namely: 265,4 (*Dual*), 265,2 W (*Single*), and 264,8 W (*S-as-D*). In addition, Figure 7.3 shows that, after completing a run of S^A, *SRVR* did not always reached it's minimum power consumption before the next job was initiated. Since all the general performance counters appear to be idle during base intervals, we assume that the cause for this is the specific cooling system implemented at *SRVR*, which often remains active while its workload has already decreased significantly. By using a single $P(\mathrm{S}^B)$ value per configuration and, thus, averaging over multiple series, we attempted to minimize the impact of this (less than optimal) behavior.

Table 7.3: The energy [left] and power [right] values of *DOCGEN*

	Energy values (Joules)			Power values (W)		
Config.	$E(\mathrm{S}^A)$	$E(\mathrm{S}^B)$	$E(\mathrm{A}_S)$	$P(\mathrm{S}^A)$	$P(\mathrm{S}^B)$	$P(\mathrm{A}_S)$
Dual	630.826	624.781	5.782	268,0	265,4	2,6
Single	682.260	664.817	17.560	272,2	265,2	7,0
S-as-D	625.730	620.496	4.977	267,1	264,8	2,2
$\sigma/Mean$	0,0395	0,0313	0,6093	0,0083	0,0009	0,5529

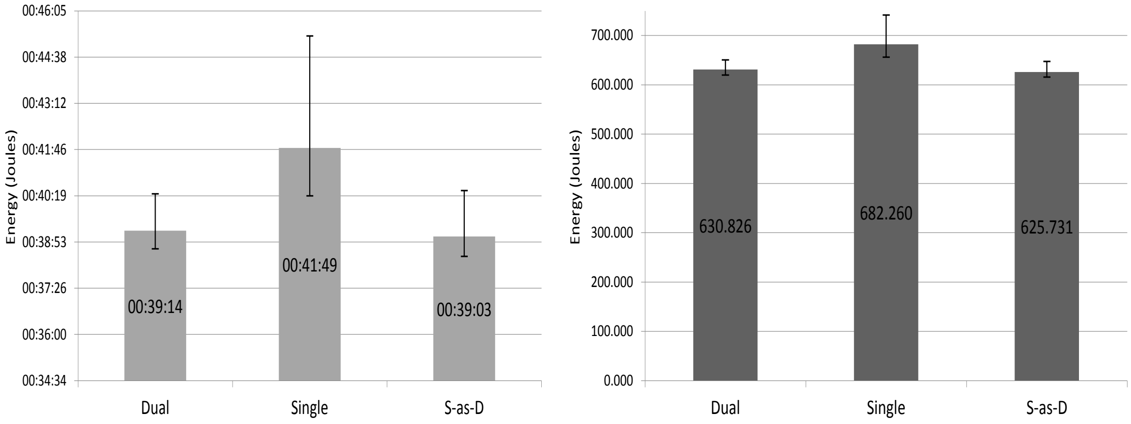


Figure 7.4: Overview of required time [left] and energy [right] per *DOCGEN* configuration

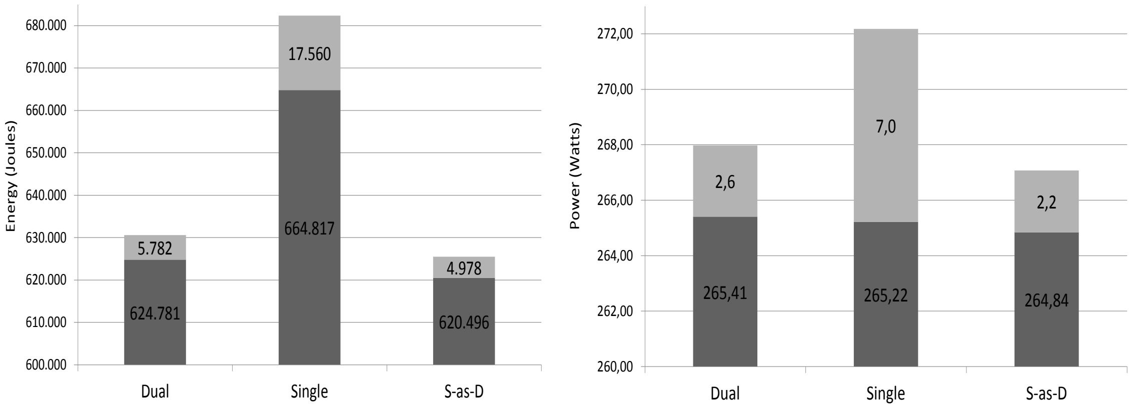


Figure 7.5: Overview of Base/App-distribution at energy [left] and power [right] values

Table 7.3 depicts the specific operational energy use values that resulted from applying the subtractive metric on the *DOCGEN* measurement data. The values show that, besides that *S-as-D* remains similar to *Dual*, there remains a clear difference between the *Dual* and *Single* configurations. Moreover, this difference is confirmed by the deviations over each column (i.e. $\sigma/Mean$), which are larger at $E(\mathrm{A}_S)$ and $P(\mathrm{A}_S)$ than at $E(\mathrm{S}^A)$ and $P(\mathrm{S}^A)$, respectively. To further illustrate the difference between the configurations, several graphs are shown in Figure 7.4 and Figure 7.5. It should be noted that at three of these graphs the y-axis does not begin at zero, because otherwise it would be difficult to notice differences between configurations. The first two graphs show clearly that the amounts of execution time and energy required at the *Single-core* configuration are more than at the other two configurations In addition, the included *error bars* indicate that the differences between the related minimal and maximum values are larger. Further, the second pair of graphs illustrate the (average) application and base ratios within the total system's energy and power consumption, respectively. With respect to the energy consumption, we see that the base at *Single* is considerably larger, which is mainly caused by the longer execution time related to this configuration. Then, with respect to the power consumption, we see that the power related to the application, i.e. $P(\mathrm{A}_S)$, of *Single* is higher than the other two configurations. We assume that this is caused, amongst others, by the higher average workload at the CPU component during S^A. Thus, because of its longer execution time and higher $P(\mathrm{A}_S)$ values, we can conclude that the operational energy use of *Single* is higher than that of *Dual.* Since both configurations process the same amount of work, we can argue that (on *SRVR*) *Dual* is more energy efficient than *Single.*

7.3.3 Applying the Power Use Range on *DOCGEN*

The final metric to be validated is the position of the configurations within the power use range of *SRVR*. Only, the main value of this metric would be to compare the positions of one or more configurations within the ranges of different systems, which was unfortunately not possible yet. However, we can still investigate how the specific $R(A_S)$'s correlate with the %CPU Time values.

Table 7.4: Overview of power use range values and %CPU Times for *DOCGEN*

Config.	$P(\mathrm{S}^R)$ (W)	$P(\mathrm{A}_S)/P(\mathrm{S}^A)$	$P(\mathrm{A}_S)/P(\mathrm{S}^R)$ :: $R(A_S)$	%CPU Time(S^A)
Dual	22,69	0,96%	11,32%	13,8%
Single	22,88	2,56%	30,41%	28,9%
S-as-D	24,26	0,84%	9,21%	11,0%

As Table 7.4 shows, the correlation between the $R(A_S)$, i.e. $P(\mathrm{A}_S)/P(\mathrm{S}^R)$, and %CPU Time values is (again) stronger than the correlation between $P(\mathrm{A}_S)/P(\mathrm{S}^A)$ and %CPU Time. Moreover, we see a slightly stronger similarity than we saw at *Systester* data with *SRVR*. We assume that the main cause for this is the difference in workload and, thus, the energy consumption profiles. For instance, *Systester* shows stable yet maximum workload at a single component, while *DOCGEN* shows more variability in the intensity at the CPU component. In addition, we expect that, in the case of *DOCGEN*, some of the other components show an increased activity during S^A as well.

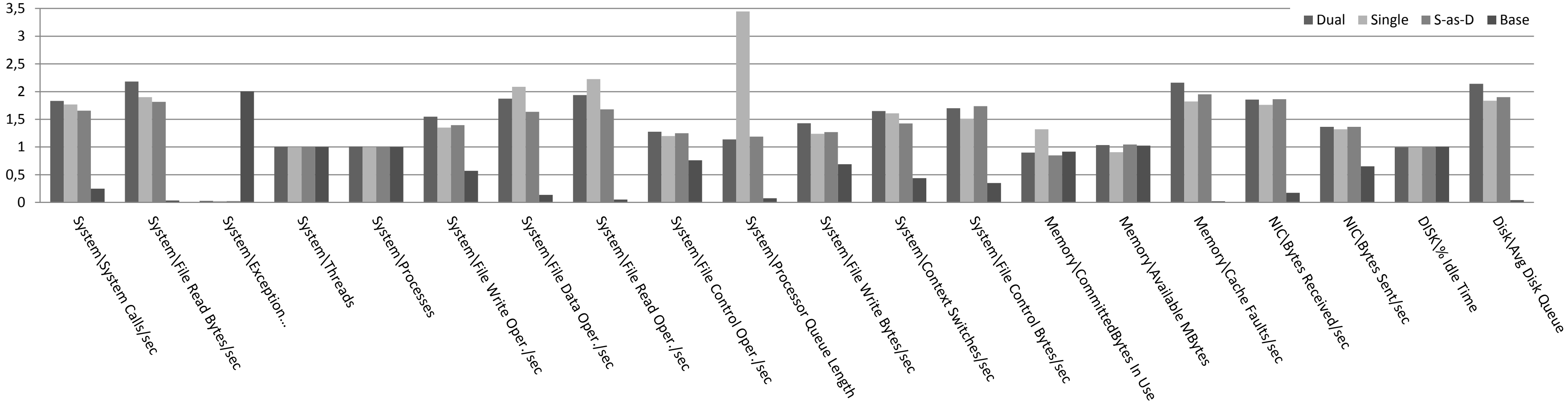
7.3.4 Analyzing the Performance of Other Components

The results in Table 7.4 show that our initial assumption with respect to the existence of a (nearly) linear dependency between the %CPU Time and power use range of a system (during the execution of an application), does hold for *DOCGEN*, to some extend. However, statistics have indicated (again) that the correlations are not (yet) significant. Therefore, we want to investigate if there are non-CPU components that are utilized considerably during the execution of *DOCGEN*. To conduct this investigation, we included the performance counters values of these other components.

Table 7.5: Overview of workload at other components during measurements with *DOCGEN*

Comp.	SYSTEM												MEMORY			NIC		DISK	
Config.	System Calls/sec	File Read Bytes/sec	Except. Dispatch/s	# Threads	File Write Operations	File Data Operations	File Read Operations	File Control Oper.	Processor Queue L.	File Write Bytes/s	Context Switches/s	File Control Bytes/s	% Comm. Bytes	Available MBytes	Cache Faults/sec	Bytes Rec./sec	Bytes Sent/sec	% Idle Time	Avg. Queue Length
Dual	12.936	113.163	0,01	432	44,1	336	292	607	0,23	115.189	1.177	65.297	10,1	3.230	40,8	61.821	120.733	98,8	0,01
Single	12.471	98.633	0,01	430	38,5	374	336	570	0,69	99.879	1.149	58.034	14,8	2.825	34,4	58.613	116.870	99,0	0,01
S-as-D	11.688	94.213	0,01	430	39,7	293	254	595	0,24	102.213	1.018	66.724	9,52	3.261	36,8	62.027	120.784	98,9	0,01
Base	1.746	1.753	1,0	430	16,2	24,2	7,9	362	0,00	55.521	313,4	13.464	10,3	3.199	0,4	5.759	57.736	100	0,00

Figure 7.6: Performance values per counter normalized to the mean over the specific counter



During the measurements with *DOCGEN*, we monitored the performance through multiple data collectors, i.e. an “overview” collector that combines the main counters of all the components and multiple “detail” collectors that each focus on a specific important component and include all counters of the specific component. The main purpose of this approach was to have a primary collection of combined performance data (e.g. compact and thus easier to process), and a secondary collection, that could be consulted when additional details are desired or during future research.

Table 7.5 (see previous page) gives an overview of the performance counters that are included in the “overview” collector besides the already discussed main CPU and Processes related counters. Each column in the table contains a specific counters while multiple counters are grouped by their component, i.e. System, Memory, Network Interface Card (NIC) and Disk. With respect to the rows, the values are grouped per configuration while the Base interval values are grouped together.

Moreover, the values in Table 7.5 indicate the workload at the different components in the case that the system is idle in comparison to when it is executing the application, but also between the different application versions. For instance, we can see significant increases at 10 of the 13 System counters, during S^A compared to the base intervals. Furthermore, the Memory, NIC and Disk also show additional activity, although at some counters there is only a marginal increase. In addition, at some of the counters, the base values are higher than the application values. For instance, this holds for *System/Exception Dispatches/sec*, which can be related to possible interferences inflicted during the preparation of the application and a new job, and *Disk/%Idle Time*, which is not unexpected because at this counter the “minimum” and “maximum” are switched (e.g. in comparison to other counters). Then, with respect to the comparison between different versions, we see that the *Single* version appears to result in more committed memory, less network traffic, and slightly less disk utilization. Finally, we also see some remarkable things with respect to the values related to *S-as-D*, for instance, at multiple counters its values are more similar to the *Single* configuration than to *Double*, despite that the reverse has been observed at the CPU and Processes counters. For this reason, we decided to include the *S-as-D* runs as a separate configuration e.g. until future research might provide a better understanding of why these specific runs occurred.

To further illustrate the relations between the different configurations and the base, the previous page also includes a graph, which shows the normalized performance values. The normalization is achieved by calculating the mean over the values per performance counter, i.e. the “counter mean”, and then dividing each counter value by its related counter mean. The result of this is that values that are similar to the counter mean receive a y-value of 1 in the graph and, also, the difference in scale between counters is minimized, which makes comparing them more practical.

7.3.5 Adjusting to the Profile of *DOCGEN*

The analysis of the other components have showed that there is indeed additional activity at the other components during the execution of *DOCGEN*. Some of these differences are only marginal (e.g. Disk) and, therefore, their influence on the power consumption seems insignificant. However, given the specific workload profile of *DOCGEN* (i.e. a high intensity beginning and a low intensity ending) and the relatively long execution time of the selected job, we should consider analyzing the two parts separately. The main reason for this is that, currently, the small differences - with respect to the system workload and the resulting power use - between the low intensity ending part and the base intervals overwhelms, or skews, the relatively large difference that is present during the high intensity beginning part due to the longer duration of the ending part, which is at least 500% longer and 125% longer in the case of *Dual* and *Single*, respectively.

To analyze these *High* and *Low* parts, we selected two measurement series of the *DOCGEN* validation data collection: a *Dual*, containing six clean runs, and a *Single*, which also contains six clean runs. For each run, we determined the duration of the *High Intensity* part, which was mainly based on the % CPU Time value of the *BriefServer* process. However, we also considered the % CPU Time values of the processors and the other processes, when the *BriefServer* process values did not indicate a sufficiently clear transition. The next step was to gather and process the WUP and PerfMon data with the new intervals, which resulted in the collection of tables and graphs displayed at the subsequent two pages.

The organization of the tables and graphs are as follows. Table 7.6 and Table 7.7 show the main (e.g. CPU and Process) and other components measurement results, respectively. Since we only included the *Dual* and *Single* version, these tables consist of 4 different "part/version"-rows. Figure 7.7 consists of 2 graphs concerning the %CPU Time values per part/version, which show the workload per CPU and per different process during the execution of *DOCGEN*, respectively. Then, Figure 7.8 also consists of 2 graphs, which concern the determined power distribution per different part/version (e.g. how much belongs to the base and the application) and a comparison between the $R(A_S)$ values of the *High* (intensity) parts and the related total %CPU Time values. With respect to the latter graph, we did not include the *Low* (intensity) parts because we wanted to limit the scope the most influential parts (e.g. $P(\mathrm{A}_S)$ of *Low* parts are less than 1 W). Finally, Figure 7.9 gives a normalized representation of the workload at the other components during the *High* parts. By normalizing the values, we can combine the counters in one (clear) graph, which makes the process of comparing the differences between the versions per counter more practical.

As we explained before, the included normalized graph should be analyzed with care, because the counters that have absolute values close to zero result more quickly in large differences in comparison to the counters that have greater absolute values. For instance, the last counter, i.e. *Disk/Avg. Queue Length*, indicates that the two versions differ by 1,2; this is the largest difference among the counters. However, the absolute values for this counter are only 0,08 and 0,02 at the *Dual* and *Single* version, respectively. Moreover, the counter Memory/Available MBytes, gives a normalized difference of 0,13, while in absolute values it appears that the *Dual* version requires more than 400MB additional memory on average. This was also visible in Table 7.5 (e.g. without separated parts), only now we can see that there is a large difference at the *Cache Faults/sec* counter between the *High* and *Low* parts, but also between the *High* parts of the two versions. Thus, besides the observed difference in the amount of occupied memory per version, which might not necessarily result in different power consumptions, we also see differences in the amount of activity at the Memory component between both versions and parts. The same holds for the %Idle Time at the Disk component, for which we already saw a small difference between the different versions, only now we see that during the *High* part of the *Dual* version the disk is almost 5% less idle.

Since the Disk and Memory components are (slightly) more active during the *High* part of the *Dual* version, we would assume that the similarity between the related $R(A_S)$ and %CPU Time metrics is weaker in comparison to the *Single* version. Figure 7.8b shows that the differences between these two metrics are 2,8% and 1,7% in the case of *Dual* and *Single*, respectively. Thus, our assumption appears to be true, despite that these values are just 1,1% apart from each other.

In conclusion, the additional analysis of different parts within the execution of *DOCGEN* has given more insight into the workload at different components. For instance, before the analysis, we assumed that the two versions produced a similar workload at the Disk component and that the 0,2% difference at the *Disk/Avg. Queue Length* was caused by noise. However, by focusing on the *High* part, we see that the two versions differ by almost 5% at this counter, which is a sizable amount that is less likely caused by noise. Therefore, besides the influence of the CPU and Memory components, the exact influence of the Disk component should also be considered within the profile of *DOCGEN*. Yet, despite the additional activity at the non-CPU components during *Dual* and the higher total %CPU Time of *Single*, we see in Figure 7.8a that *Single* appears to have a greater $P(\mathrm{A}_S)$ value. Consequently, we can argue that - within our setup - the specific workload at the CPU component has the most influence on the power consumption during the execution of *DOCGEN* and, thus, optimizing the workload at this component would be a smart decision.

Table 7.6: Overview of the main measurement results per part/version of *DOCGEN*

	#	Duration (avg.)		$E(\mathrm{S}^A)$ :: Energy Use (Joules)						$P(\mathrm{S}^A)$ :: Power Use (W)	
Part :: version	Runs	Seconds	$\sigma(T)$	Mean	Median	Min.	Max.	$\sigma(E)$	σ/Mean	Mean	Mean(σ p/run)
High :: Dual	6	288	22,06	79.418	78.372	72.493	88.166	6.240	0,0786	275,9	1,43
High :: Single	6	1.041	49,37	289.112	289.191	268.556	308.080	13.717	0,0474	277,8	1,33
Low :: Dual	6	2.071	33,76	551.575	550.628	538.788	564.979	9.081	0,0165	266,3	0,63
Low :: Single	6	1.402	40,86	372.771	370.031	360.489	393.653	11.043	0,0296	265,9	0,51

Table 7.7: Overview of workload at other components per part/version of *DOCGEN*

Comp.	SYSTEM												MEMORY			NIC		DISK	
Part	System Calls/sec	File Read Bytes/sec	Except. Dispatch/s	# Threads	File Write Operations	File Data Operations	File Read Operations	File Control Oper.	Processor Queue L.	File Write Bytes/s	Context Switches/s	File Control Bytes/s	% Comm. Bytes	Available MBytes	Cache Faults/sec	Bytes Rec./sec	Bytes Sent/sec	% Idle Time	Avg. Queue Length
H :: D	42.856	424.806	0,01	442	231,8	1.841	1.609	959	0,75	346.007	5.773	160.127	8,69	3.345	251,6	132.937	174.669	93.0	0,08
H :: S	19.237	152.982	0,01	430	66,85	826,2	759,4	590,1	1,20	124.625	2.129	65.843	13,6	2.928	66,8	70.140	124.615	97,8	0,02
L :: D	7.854	62.224	0,01	431	18,2	56,9	38,4	556,7	0,10	84.270	463,8	63.800	9,43	3.282	12,6	51.882	114.157	99,6	0,00
L :: S	7.901	61.689	0,01	426	18,1	56,4	38,3	550,2	0,14	82.278	451.7	54.114	13,64	2.924	11,6	52.334	113.666	99,8	0,00

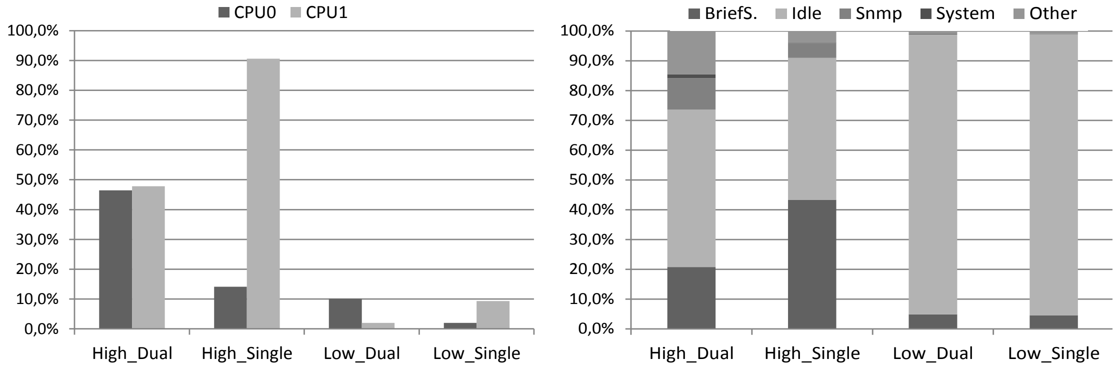


Figure 7.7: %CPU Time per part/version of *DOCGEN* per CPU [left] and Process [right]

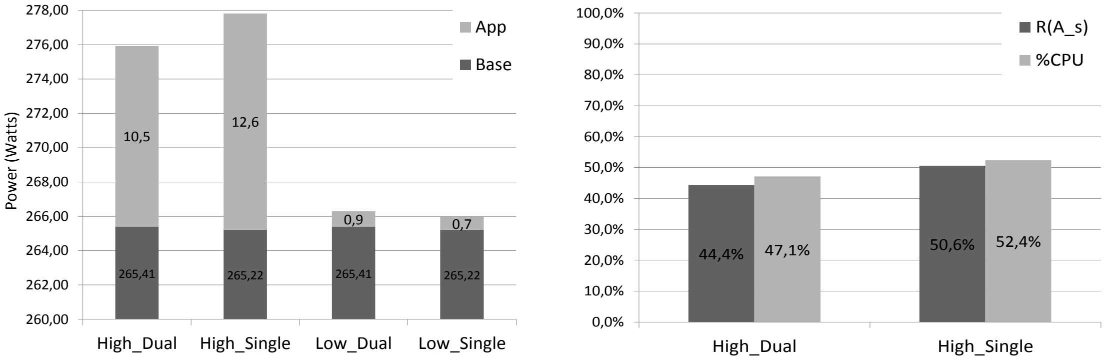


Figure 7.8: Overview of Power per part/version of *DOCGEN* [left] and Ratio's [right]

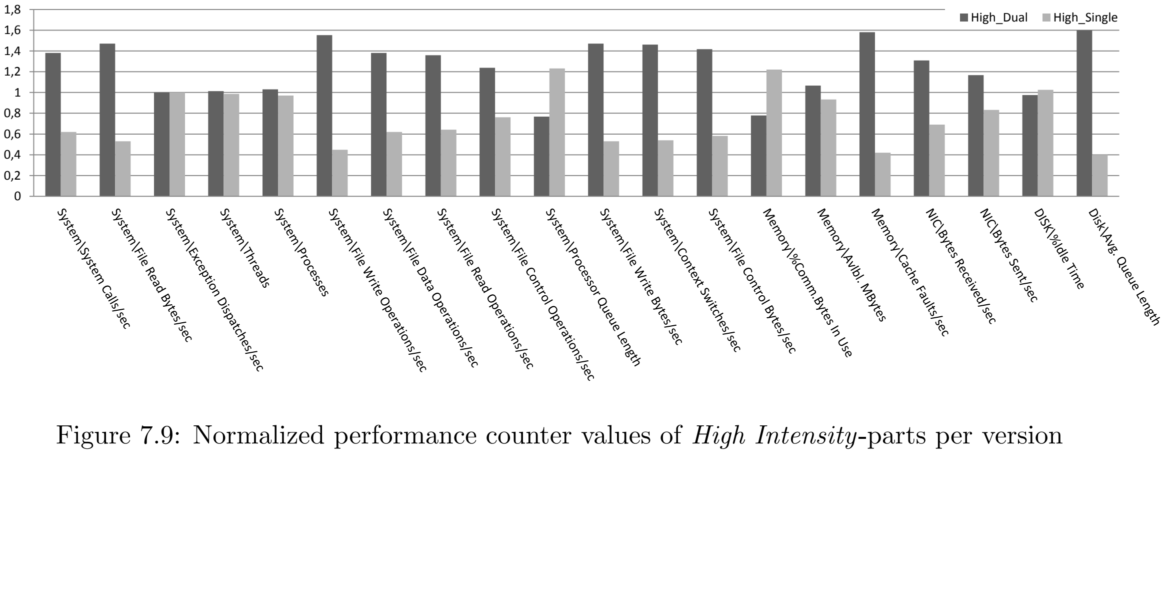


Figure 7.9: Normalized performance counter values of *High Intensity*-parts per version

Chapter 8

Discussion

The following chapter provides a general overview of the findings and research contributions of this thesis. Furthermore, we will elaborate on their significance and the influence of limitations.

8.1 Findings & Research Contribution

The following items are ordered by their importance within the thesis project. We begin with our method for measuring the power and workload of systems during the execution of an application. Subsequently, we discuss the different metrics that can be applied on the clean measurement data.

8.1.1 Method for Measuring the Power and Workload of Systems

By now, it should be clear that it was not at all an easy task to create the reliable datasets that were necessary for investigating the metrics. Yet, despite the still experimental nature of this research, we made the utmost effort of considering all the important factors in order to maximize the reliability of the research. Consequently, our method provides extensive guidance for coping with different computer systems, variable background workloads, relatively reliable instrumentation, and inflicting minimal interferences by means of conducting measurements remotely. In addition, we explain how the measurement data of PerfMon and WUP can be combined and, subsequently, how the intervals related to execution of applications can be interpreted. Finally, we have showed how the performance data can be used to analyze the workload at different components and how unclean runs, or characteristic, can sometimes be linked to certain components or counters.

To our knowledge, important aspects such as the extensive process of creating reliable datasets, considering properties of different systems, and analyzing the characteristics (and possible influences) of system components during measurements are yet underexposed in most other research.

8.1.2 Determining the Operational Energy Use of an Application

Most of the related research mainly focuses on the comparison of total system energy use values to compare (e.g. the efficiency of) different applications, or versions. We argue that metrics based on these values can still provides useful insights. First, there is the metric to compare different application versions on a single system. The results from this metric enable us to determine which version is more energy-efficient, on the basis of a particular system. Second, there is the metric of comparing a single application version on different systems. This would then enable us to determine which system is more energy-efficient, on the basis of a particular application.

However, given that product software is often executed on many different systems (e.g. with various properties), assumptions made with the former metric can be difficult to generalize and the latter metric is difficult to interpret when systems have power use values on different scales.

In this research, we have demonstrated a subtractive method that can be used for determining the energy use solely related to S^A, or the operational energy use. We argue that this methods provides more generalizable and more easier to interpret results, by excluding the Base power consumption of systems, which we assume (after extensive analyzes) is unrelated to S^A. Comparisons of the operational energy use values show that the similarities between different systems are considerably stronger than in comparison to the total energy values of the systems during S^A. However, exact energy consumption values that are required to execute a specific application independent of the underlying system have not yet been found with this metric.

8.1.3 Benefits of the Power Use Range

This research showed that, to cope better with different power consumption scales of systems, we require to determine the power use range of systems, which is based on the $P(\mathrm{S}^B)$ and $P(\mathrm{S}^M)$ values. By doing so, we saw that, although the size of the range can be different between systems, most systems share a considerably strong relation between the position of the power consumption solely related to the execution of the application, i.e. $P(\mathrm{A}_S)$, within the power use range and the amount of workload at the CPU component, i.e. the main "energy consumer" in most systems.

This observation opened the way of investigating the "prediction" metrics introduced in Section 5.2.2. However, since statistics showed that the correlation between the variables were not significant and the fact that we had difficulties with determining the exact $P(\mathrm{S}^M)$ values, we did not pursue making any real conclusions for these metrics at the moment. Hence, future research should focus on a method for determining reliable $P(\mathrm{S}^M)$ values before considering these metrics.

8.1.4 Architectural aspects

Since eventually the major part of this thesis is about ensuring reliable measurements (due to i.a. the unreliability of *Joulemeter* and variability of systems), a smaller than originally planned part is about the influence of architectural aspects on the sustainability of software. Yet, we managed to include at least one aspect, i.e. executing an application by a single core vs. multiple cores. This effort proved to be a valuable one, because our data shows that the CPU component has the most influence on the power consumption at both of the applications tested within this thesis. We also concluded that multi-core processing is in fact more energy efficient in practically all of the cases. Only with *SRVR* and *Systester* is the single-core version of *Borwein* slightly more efficient, yet this may also have been caused by the underlying virtual machine or the type of memory in this particular system. With *DOCGEN*, the dual-core version is three times more energy efficient.

8.1.5 Current practices studies

This research showed that before 2014 there was hardly any literature that combines the research fields of Software Architecture and Green Software. Furthermore, our brainstorm illustrates that software architects have various ideas concerning aspects that might influence the sustainability of software, yet they have no methods or tools to actually measure or (effectively) influence it.

8.2 Research Limitations

Despite our best efforts, there are various limitations to the research and the conducted experiments. In the following section, we will discuss the most important of these limitations.

8.2.1 Influence of Other Components

As we explained before, we deliberately used *Systester* as the initial test application, due to its "simple" workload (e.g., mostly uses the CPU, plus some Memory in the case of the dual-core version). In reality, an application often produces less constant workloads, uses multiple processes and it can also produce workloads at other components than the CPU. As we saw with *DOCGEN*, workload can also be generated at the Disk component (e.g. this creates the necessity to clean the disk and cache at each experiment run) or, for instance, in a "distributed" network environment.

8.2.2 Measurement Instrumentation

We realize that since the WUP meter and PerfMon have a granularity of one measurement record per second while systems process millions of instructions per second, we miss a considerable amount of detail. Furthermore, the WUP and PerfMon logs are sometimes corrupted: the chance of zero differences between the PerfMon and WUP timestamps is 0% at longer measurement-logs (e.g., >4 hours). In addition, some values in PerfMon logs are higher than their maximums in reality.

8.2.3 Cooling-down after a Measurement Run

The application execution time intervals of *Systester* and *DOCGEN* were determined based on the activity of the "Systester" and "BriefServer" processes, respectively, which means that we only included the moments in which the processes had a %CPU Time value greater than zero. Although this was the best method available for methodically collecting the measurement data, the disadvantage of this was that we did not include the "cooling-down" period, which often occurred after the execution of A. During this period, the power use of the system appeared to be considerably higher than its $P(\mathrm{S}^B)$ value, despite that the execution of A was already finished.

The obstacle that we faced was that this specific period did not have a constant duration, hence it was impossible to collect the values methodically. Furthermore, we can not reliably determine which portion of this additional power is actually related to S^A. Therefore, we ignored these portions completely. Also, a similar event sometimes occurred before S^A; the system energy use would rise although S^A had not been started, or the %CPU Time of the process was still 0%.

NB. for the determination of $P(\mathrm{S}^B)$ values, we did reckon with the presence of cooling-down periods, by including a buffer of at least 30 seconds between each Application and Base interval.

8.2.4 Influence of Climate on the Efficiency of Components

The amount of power consumption of the CPU component can be influenced by several factors, including: dynamic power consumption, short-circuit power consumption, and power loss due to transistor leakage currents. The amount of transistor leakage can be an important aspect since it influences the efficiency of a system's performance and it tends to inflate for increasing temperatures. Hence, we considered briefly the influence of the Thermal Design Power (TDP), which is the maximum amount of heat generated by the CPU that the cooling system is required to dissipate.

During our investigation, we observed that the power consumption of some of the systems, running on their maximum performance and processing a maximum workload, kept on increasing over a longer period of time while the power consumption at systems with a limited performance would reach their maximum value almost instantly. In addition, we observed at a specific system that when we removed the side panel of the case (e.g. improve heat dissipation), its $P(\mathrm{S}^A)$ value decreased with 5 Watts. This made use realize that the actual room temperature as well as the system cooling performance should also be controlled, or at least monitored, during measurements.

8.2.5 Concerning *DOCGEN*

As we discussed in Chapter 7, conducting measurements with *DOCGEN* was not straightforward. However, although the measurements did not went perfectly they were far from complete failures. In fact, by documenting our experiences (including pitfalls), we can make better decisions in future research. Hence, the main limitations concerning measurements with *DOCGEN* are as follows:

Sporadic increased base power consumption - Both with *Systester* and *DOCGEN*, we saw that system *SRVR* sometimes had an increased base power consumption over remarkably lengthy periods of time. Despite additional analysis, the exact cause for this behavior still remain inexplicable. Yet, we presume that it is caused by *SRVR*'s outdated cooling-system.

Unpredictable *S-as-D* runs - Our method for adjusting the CPU affinity of an application (e.g. to simulate a single-core version), did not prove to be reliable with *DOCGEN*. Therefore, we should investigate how we can improve its reliability or consider modifying the source code.

Increasing execution times after multiple runs - Within our dataset, we saw that at some series the execution time per run increased during the series. Thus, instead of a benefit of e.g. caching, we see a disadvantage when multiple runs a combined within a series. Since we do not know the exact cause of this, we can only advice to consider additional removal of cached or result data and possible reboot the system between two runs (NB. this does increase the length of measurement series and it most likely causes additional interferences).

No automated (remote) execution method - Since we were not able to operate *DOCGEN* from command-line, it is more challenging (or possibly impossible) to initiate multiple series of runs, where each run lies within a similar environment. We aimed to minimize and standardize the amount of interference, yet in the future more automation should be considered.

Possible influence of virtual machine - We realize that *SRVR* was not a real dual-core system, despite that the virtual machine layer limited its amount of cores to 2 and that we had exclusive access to the system. Given our limited experience with VM's and server, we initially settled with this system, yet in retrospect we should have considered a different one.

Test version of *DOCGEN* - Due to a limited amount of resources available for deploying *DOCGEN* in our environment, we worked with a version that had some defects. For instance, when a job was finished, the generated documents (in PDF format) were not transfered to the client system. However, since our focus was on the CPU component (e.g. generating the documents from the definitions), we settled with this defect. Nevertheless, we should ensure a defect free version of *DOCGEN* in order to produce better measurements in the future.

8.2.6 Other limitations

State of the systems - During each series of runs, we recorded the *System Up time* of the system and (at most systems) we also performed a system reboot before a series. As we explained before, we observed that, on average, systems require around 15 minutes to settle after a reboot. With respect to the *System Up time*, we did not see remarkable differences between a run that was executed at 20 minutes *System Up Time*, or at 200 minutes. However, we did see that after the 15 minutes of settling from the reboot, the system would stay idle for around 30 minutes and after that it would have an increased power use for a sizable long interval. We did not retrieve the cause for this increase, yet we presumed that it may be a Windows-related process that would be triggered after a specific amount of idle time.

Peripherals - We excluded the power use of peripherals e.g. by disconnecting the USB devices. We did not measure the power use of displays, because the WUP can only measure 1 device. The benefit of *Systester* is that it does not produce any visual results and, thus, we can omit the peripherals. Yet, with a more complex application, their influence might not be ignored.

%CPU Time: User/Privileged - During our quest for a sufficiently detailed and reliable performance measurement tools, such as PerfMon, we also considered the distribution of different modes within the %CPU Time, namely: *User* and *Privileged* mode, with *User* mode being the processing mode in which applications run and *Privileged*, or *kernel*, mode the processing mode that allows code to have direct access to all hardware and memory in the system. The reason for why this should be considered (in the future) is that the %CPU Time values at the Process level are often dependent of the %User Time at the CPU level.

Chapter 9

Conclusion

Now we have come to the final chapter of this thesis document, which includes a general conclusion on the research and the thesis project, an overview of the thesis results linked to the research questions, and, last but certainly not least, we discuss various opportunities for future research.

9.1 General Conclusion

With this thesis project, we envisioned to bridge the gap between the fields of Software Architecture and Green Software, i.e. how the former can be used to realize aspects of the latter. However, the immaturity of the latter field (e.g. absence of adequate instrumentation) forced us to focus our study on the process of developing proper measuring methods as well as intuitive metrics for determining the *operational energy use* of applications. Due to this change of plans and the desire to perform a case-study at the research company, the time to finish the project has been increased substantially. Nevertheless, we argue that it is, above all, important that we have overcome many obstacles during this project in terms of collecting empirical data, considering various aspects, and of course conducting valid research. With respect to architectural aspects, we have shown in detail why a dual-core version is in most cases more energy-efficient. Thus, the next step would be making a trade-off between the additional amount of developing time and the potential operational benefits. With respect to monetary implications, we see that the dual-core version of *DOCGEN*, i.e. generating 5000 "Huurverhoging" documents, requires on average 11.778 Joules less than the single-core version. Hence, if this job would be executed a million times, it saves 3.271 kWh, which results in a monetary saving lying between €650 and €750 (e.g. depending on electricity pricing).

9.2 Thesis Results Linked to RQs

The purpose of this section is to match the research question, introduced in Section 1.3, with a list of brief answers that refer to the corresponding chapters, containing more elaborate explanations.

RQ. **How can we determine the operational energy use of application software?**

The main deliverable of this research is our method for measuring the power use and workload of systems during base states and the execution of an application. We showed in Chapters 6 and 7 that by combining this data, we can more reliably and less system dependent determine the amount of energy required to execute an application. However, we also showed that we are not (yet) able to determine completely system independent values. Therefore, the answer to the main research question is that subtracting the base power consumptions results in more representable operational energy use values, yet the influence of other system properties should also be considered.

SQ. 0 *What are the current practices, both theoretical and practical, with regards to software architectural aspects for measuring and developing sustainable software?*

Chapter 4 showed that the body of relevant literature is small, yet growing, and that software architects have no methods or tools to actually measure or influence the sustainability of software.

SQ. 1 *How can we reliably measure the energy use of a computer system, execute an application and concurrently monitor its workload, with minimal interference?*

In Chapter 4 we discussed our considerations concerning the reliability of instrumentation for measuring the power consumption and workload of systems. We have concluded that to remotely measure and monitor a system, the WUP meter together with PerfMon is the best combination.

SQ. 2 *How can we isolate the energy use solely related to the execution of an application (e.g., perform a specific task) from the measured energy use of the underlying system(s)?*

In Chapter 6 we explained how we used different performance counters to determine whether during an Application run the workload at the systems was mostly created by the specific application created. If this was the case, we argued that the additional power use in comparison to the system's base state, was inflicted solely by the application. Subsequently, we determined the energy use solely related to the application by subtracting the amount related to the base state.

SQ. 3 *How can we use system specific properties to normalize the measured energy usages of executions of a single application on different systems and make them comparable?*

In Chapters 6 and 7 we showed how we determined the power use range of each system and why this metric can be useful when systems would have power use values on different scales.

SQ. 4 *How do the metrics behave in environments with other applications and/or systems?*

In Chapter 6 we discussed various validity aspects of the metrics, such its rigidity. For instance, we investigated differences between different versions between the systems, which proved to be remarkably stable. Further, in Chapter 7, we discussed the case-study that we performed with *DOCGEN*, which provided measurements within a secondary environment and valuable insights.

SQ. 5 *To what extend can the resulting metrics be used to determine the influence of software architectural aspects on the sustainability of software?*

Although we have not yet succeeded in developing completely system independent metrics, we do now have metrics that provide more precise results (e.g. more reliably measured and less unrelated data). Therefore, we were better able to compare difference between application versions, such as single vs. multi-core processing or the Gauss-Legendre vs. the Borwein algorithms.

9.3 Future Research

Besides that this research has provided a extensive collection of new insights, it has also produced many new opportunities for future research work. Hence, we discuss a few of these opportunities.

Including other task sizes - During this project we only considered a task in *Systester* of calculating $8 \cdot 10^6$ Pi decimals. We can extend our research by choosing longer tasks, such as $32 \cdot 10^6$ and investigate the relations between the two sizes. Furthermore, we can extend the durations of base intervals between the measurements runs. Both these changes result in longer intervals, which (presumably) further improves the quality of the measurement data.

Instrumentation - Given the facts that WUP meter is not very cheap and that it requires to be placed between the Application system and the power source, we might consider replacing it with *Joulemeter*. Now that we have developed our own set of metrics, we could perform additional measurements during which we use both a WUP and Joulemeter concurrently and/or solely *Joulemeter* (e.g. with or without calibration by the WUP). Subsequently, we could compare the different measurement data and investigate to what extend there are similarities. However, a few challenges concerning the quality of the measurements remain. For instance, because *Joulemeter* needs to be executed on the Application system, we cannot initiate or measure with it without creating additional interference at the Application system.

Control the power use range artificially - One of the most interesting aspects that could be investigated is the influence of modifying a system's base power use or its maximum power on the amount of energy required solely for S^A. We can control both values as follows: the base power use can be increased by creating artificial workload with *Pressure* (cf. Section 6.4), and the maximum power use can be altered by modifying the performance of the system (e.g. locking the CPU frequency). It would be interesting to study by what degree the amounts of required execution time and energy will change when, for instance, the workload at the CPU component is artificially raised to 50% or when its performance is decreased by 50%. Other research has showed that (certain) systems work significant less efficient above a certain percentage of workload e.g. we can investigate if the same holds with our configuration (NB. we have concluded that *Systester* is not the right application for this kind of investigation).

Other processor types - Since modern computers often have more than 2 CPUs, the research can be extended by including system with 4 or 8 CPUs, and comparing the results with our dual-core systems. Also, we have conducted measurements with a single-core system and with two dual-core systems with Intel's Hyper-Threading Technology (e.g. having 4 logical processing cores). Although this gave us some interesting insights, it also provided too many challenges for the moment and, thus, we only included the dual-core systems in this research.

Other operating systems - The research can be extended by performing similar measurements with systems that run on other (potential more energy stable) operating systems, such as Mac OS or Ubuntu. However, additional resources are needed to realize this, such as experience with the operating systems and, most importantly, we need to have suitable alternatives for the tools used during the experiments, such as *Pressure*, *ThrottleStop*, and *HeavyLoad*.

DOCGEN ::

Multiple clients - During our case-study, we only considered measuring the power consumption of *SRVR* while "serving" a single client. In the real world, the system is designed to process jobs from multiple clients concurrently. Therefore, the challenge in future research, is to create a test environment in which multiple clients create *DOCGEN* jobs that need to be processed by *SRVR*. This way, we gain more realistic insights into to what extend a multi-core processing version is beneficial over a single-core version. However, if we strive for a more realistic environment, we should also deploy the application on a more modern test system with better specifications.

Possible influence of other systems - Since *DOCGEN* requires at least one additional system, i.e. the Oracle DB, future research might consider isolating this system as well. In addition, all of the related systems (including clients) could placed in an exclusive environment. This way, we can ensure better that the present workload is inflicted by *DOCGEN* and, thus, the interference during measurements will be minimized.

Affinity of other processes - We mentioned that we only controlled the Affinity of the *BriefServer* process, yet we can also consider investigating the affinity of other processes.

Architectural aspects - Given our positive initial results with *DOCGEN*, concerning the difference between single and dual-core processing, we can consider modifying the source code of the application to test whether our method of adjusting the CPU affinity is reliably. Furthermore, we can consider other architectural aspects related to the design of *DOCGEN* (e.g. document generating methods), which can be altered by modifying the source code or, if possible, by adjusting its configuration after it has been compiled.

Bibliography

[1] William M Adams. The future of sustainability: Re-thinking environment and development in the twenty-first century. In *Report of the IUCN renowned thinkers meeting*, volume 29, page 31, 2006.

[2] Luca Ardito. *Energy-aware Software.* PhD thesis, Politecnico di Torino, 2014.

[3] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice.* Addison-Wesley Professional, 3rd edition, 2012.

[4] G. Bekaroo, C. Bokhoree, and C. Pattinson. Power measurement of computers: Analysis of the effectiveness of the software based approach. *International Journal of Emerging Technology and Advanced Engineering*, 4(5), May 2014.

[5] Paolo Bozzelli, Qing Gu, and Patricia Lago. A systematic literature review on green software metrics. Technical report, Technical Report: VU University Amsterdam, 2013.

[6] S.A. Broverman, Society of Actuaries, Casualty Actuarial Society, and Actex Publications. *Actex study manual, Course 1 examination of the Society of Actuaries, Exam 1 of the Casualty Actuarial Society.* Number v. 1 in Actex Study Manual, Course 1 Examination of the Society of Actuaries, Exam 1 of the Casualty Actuarial Society. Actex Publications, 2001.

[7] David J. Brown and Charles Reams. Toward energy-efficient computing. *Communications of the ACM*, 53(3):50, March 2010.

[8] Ian Burton. Report on reports: Our common future. *Environment: Science and Policy for Sustainable Development*, 29(5):25–29, 1987.

[9] Elsevier B.V. Sciencedirect. *http://www.sciencedirect.com.proxy.library.uu.nl/science/search/*, 2014.

[10] E. Capra, C. Francalanci, and S. Slaughter. Is software green? application development environments and energy efficiency in open source applications. *Information and Software Technology*, 54(1):60–71, 2012.

[11] L Dobrica and E Niemela. A survey on software architecture analysis methods. *Software Engineering, IEEE Transactions on*, 28(7):638–653, 2002.

[12] John Elkington. Enter the triple bottom line. *The triple bottom line: Does it all add up*, pages 1–16, 2004.

[13] M.A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, and J. Visser. Seflab: A lab for measuring software energy footprints. In *Green and Sustainable Software (GREENS), 2013 2nd International Workshop on*, pages 30–37, May 2013.

[14] Arlene Fink. *Conducting Research Literature Reviews: From the Internet to Paper.* SAGE Publications, 2005.

[15] Martin Fowler. Inversion of control containers & the dependency injection pattern, 2004.

[16] David Garlan and Mary Shaw. An Introduction to Software Architecture. Technical report, Carnegie Mellon University, Pittsburgh, January 1994.

[17] K. Glynn. Throttlestop 6.00. *http://www.techpowerup.com/downloads/2288/throttlestop-6-00/*, October 2013.

[18] M. Goraczko, A. Kansal, J. Liu, and F. Zhao. Joulemeter 1.2. *http://research.microsoft.com/en-us/projects/joulemeter/*, September 2011.

[19] C. Hart. *Doing a Literature Review: Releasing the Social Science Research Imagination*. Open university set book. SAGE Publications, 1998.

[20] Alan Hevner and Samir Chatterjee. *Design Research in Information Systems: Theory and Practice*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[21] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, March 2004.

[22] Christine Hofmeister, Philippe Kruchten, Robert L. Nord, Henk Obbink, Alexander Ran, and Pierre America. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software*, 80(1):106–126, January 2007.

[23] ThinkTank Energy Products Inc. Watts up? plug load meters. *https://www.wattsupmeters.com/*, 2012.

[24] ISO/IEC. ISO/IEC 25010:2011 – Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. Technical report, ISO/IEC, 2011.

[25] ISO/IEC/IEEE. Iso/iec/ieee systems and software engineering – architecture description. *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)*, pages 1–46, Dec 2011.

[26] E. Jagroep, J.M. Van der Werf, S. Jansen, M. Ferreira, and J. Visser. Profiling energy profilers, 2015.

[27] T. Johann, M. Dick, S. Naumann, and E. Kern. How to measure energy-efficiency of software: Metrics and measurement results. In *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, pages 51–54, June 2012.

[28] Eva Kern, Markus Dick, Stefan Naumann, Achim Guldner, and Timo Johann. Green Software and Green Software Engineering–Definitions, Measurements, and Quality Aspects. *on Information and Communication Technologies*, page 87, 2013.

[29] Sedef AKINLI Kocak. Green software development and design for environmental sustainability. In *11th International Doctoral Symposium an Empirical Software Engineering (IDOESE 2013). Baltimore, Maryland*, volume 9, 2013.

[30] Heiko Koziolek. Sustainability evaluation of software architectures. In *Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS - QoSA-ISARCS '11*, page 3, New York, New York, USA, June 2011. ACM Press.

[31] Patricia Lago and Toon Jansen. Creating environmental awareness in service oriented software engineering. In E.Michael Maximilien, Gustavo Rossi, Soe-Tsyr Yuan, Heiko Ludwig, and Marcelo Fantinato, editors, *Service-Oriented Computing*, volume 6568 of *LNCS*, pages 181–186. Springer Berlin Heidelberg, 2011.

[32] F Losavio, L Chirinos, A Matteo, N Lvy, and A Ramdane-Cherif. {ISO} quality standards for measuring architectures. *Journal of Systems and Software*, 72(2):209 – 223, 2004.

[33] A. Mahesri and V. Vardhan. Power consumption breakdown on a modern laptop. In *Proceedings of the 4th International Conference on Power-Aware Computer Systems*, PACS'04, pages 165–180, Berlin, Heidelberg, 2005. Springer-Verlag.

[34] Sara S. Mahmoud and Imtiaz Ahmad. A Green Model for Sustainable Software Engineering. *International Journal of Software Engineering and Its Applications*, 7(4), 2013.

[35] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decis. Support Syst.*, 15(4):251–266, December 1995.

[36] Microsoft. Windows performance monitor. *http://technet.microsoft.com/en-us/library/cc771692.aspx/*, April 2007.

[37] Microsoft Patterns & Practices Team. *Microsoft Application Architecture Guide.* Microsoft Press, 2 edition, 2009.

[38] Mark P Mills. The cloud begins with coal: an overview of the electricity used by the global digital ecosystem. Technical report, Digital Power Group, August 2013.

[39] Simon Mingay. Green IT: The New Industry Shockwave. *Gartner RAS Research Note G*, 153703:7, 2007.

[40] Jeremy Moon. The contribution of corporate social responsibility to sustainable development. *Sustainable Development*, 15(5):296–306, 2007.

[41] San Murugesan. Harnessing green it: Principles and practices. *IT Professional*, 10(1):24–33, 2008.

[42] S. Naumann, M. Dick, E. Kern, and T. Johann. The {GREENSOFT} model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, 1(4):294 – 304, 2011.

[43] Chitu Okoli and Kira Schabram. A guide to conducting a systematic literature review of information systems research. *Sprouts: Working Papers on Information Systems*, 10(26), 2010.

[44] Alex F Osborne. Applied imagination: principles and procedures of creative problem solving. *Charles Scribeners Sons, New York*, 1953.

[45] B. Penzenstadler, V. Bauer, C. Calero, and X. Franch. Sustainability in software engineering: a systematic literature review. In *16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, pages 32–41. IET, 2012.

[46] C. Pettey and R. van der Meulen. Cfo advisory: The impact of sustainability on enterprise performance. *http://www.gartner.com/newsroom/id/1749115*, July 2011.

[47] G. Procaccianti, P. Lago, and G.A. Lewis. A catalogue of green architectural tactics for the cloud. In *Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2014 IEEE 8th International Symposium on the*, pages 29–36, Sept 2014.

[48] W. W. Royce. Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[49] M. Russinovich. Pstools suite. *http://technet.microsoft.com/en-us/sysinternals/bb896649.aspx/*, May 2014.

[50] I. Sartori and A.G. Hestnes. Energy use in the life cycle of conventional and low-energy buildings: A review article. *Energy and Buildings*, 39(3):249 – 257, 2007.

[51] Chiyoung Seo, George Edwards, Daniel Popescu, Sam Malek, and Nenad Medvidovic. A framework for estimating the energy consumption induced by a distributed system's architectural style. In *Proceedings of the 8th International Workshop on Specification and Verification of Component-based Systems*, SAVCBS '09, pages 27–34. ACM, 2009.

[52] Mark Silver, Lynne Markus, and Cynthia Beath. The information technology interaction model: A foundation for the mba core course. *MIS Q.*, 19(3):361–390, September 1995.

[53] Herbert A. Simon. *The Sciences of the Artificial (3rd Ed.)*. MIT Press, Cambridge, MA, USA, 1996.

[54] Bob Steigerwald and Abhishek Agrawal. Green software. *Harnessing Green IT: Principles and Practices*, page 39, 2012.

[55] Yuzhong Sun, Yiqiang Zhao, Ying Song, Yajun Yang, Haifeng Fang, Hongyong Zang, Yaqiong Li, and Yunwei Gao. Green challenges to system software in data centers. *Frontiers of Computer Science in China*, 5(3):353–368, 2011.

[56] Mikael Svahnberg, Claes Wohlin, Lars Lundberg, and Michael Mattsson. A method for understanding quality attributes in software architecture structures. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering - SEKE '02*, page 819, New York, New York, USA, July 2002. ACM Press.

[57] Juha Taina. Good, bad, and beautiful software-in search of green software quality factors. *CEPIS UPGRADE*, 12(4):22–27, 2011.

[58] Hideaki Takeda, Paul Veerkamp, and Hiroyuki Yoshikawa. Modeling Design Process, December 1990.

[59] J. Tayeb, K. Bross, S. Chang, L. Cong, and S Rogers. Intel Energy Checker Software Development Kit User Guide. Technical report, Intel, 2010.

[60] L Tsatiris, P. Anastasiadis, and M. Kaniadakis. System stability tester: test your system's stability and performance by calculating millions of digits of pi. http://systester.sourceforge.net/, July 2012.

[61] L Tsatiris, P. Anastasiadis, and M. Kaniadakis. System stability tester: test your system's stability and performance by calculating millions of digits of pi. http://systester.sourceforge.net/, July 2012.

[62] Inge van de Weerd and Sjaak Brinkkemper. *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications*. IGI Global, July 2008.

[63] P. Verschuren and H. Doorewaard. *Designing a Research Project*. Eleven International Publishing House, 2 edition, 2010.

[64] S Vijayalakshmi, G Zayaraz, and V Vijayalakshmi. Article: Multicriteria Decision Analysis Method for Evaluation of Software Architectures. *International Journal of Computer Applications*, 1(25):22–27, 2010.

[65] T. Willhalm, R. Dementiev, and P. Fay. Intel performance counter monitor - a better way to measure cpu utilization. https://software.intel.com/en-us/articles/intel-performance-counter-monitor/, December 2014.

[66] Lai Xu and Sjaak Brinkkemper. Concepts of product software. *European Journal of Information Systems*, 16(5):531–541, Oktober 2007.

[67] Benjamin Zhong, Ming Feng, and Chung-Horng Lung. A Green Computing Based Architecture Comparison and Analysis. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 386–391. IEEE, December 2010.

List of Figures

1.1 Core activities in Software Architecture Design, according to [22] 5

2.1 Research Design: the IS Research Conceptual Framework by [21] 7
2.2 Research Model 9
2.3 Process-Deliverable Diagram 12

3.1 Venn diagram of Sustainability factors 15
3.2 The GREENSOFT model by [42] 18
3.3 Waterfall Model of the SDLC based on [48] 18
3.4 Quality model for internal and external quality by [37] 19

4.1 Structured Literature Review Approach 21
4.2 Structured Literature Review Search Terms 22
4.3 Structured Literature Review Search Terms 23
4.4 Representation of Incremental Application Versions 26
4.5 Instrumentation: a WattsUP? PRO power meter 27

5.1 Representation of measuring multiple executions of A 30
5.2 Representation of measuring with multiple *AppSys*-configurations 31
5.3 Comparing data (and results) from multiple runs with a single *AppSys*-configuration 32
5.4 Comparing results between different systems with a single application version . . . 32
5.5 Comparing results between different application versions with a single system . . . 32
5.6 Example of power distributions per system :: absolute P [left] and $R(A_S)$ [right] . 33

6.1 Representation of Experimental Setup I 36
6.2 WUP data - Series of measurement runs 40
6.3 Results - Execution time per application per system 42
6.4 Results - Energy use per application per system 42
6.5 Overview of $E(\mathrm{S}^A)$ [left] and $E(\mathrm{A}_S)$ [right] values 44
6.6 Overview of $P(\mathrm{A}_S)/P(\mathrm{S}^A)$ [left] and $R(A_S)$ [right] ratios 46
6.7 Overview of $R(A_S)$ [left] and %CPU Time(S^A) [right] ratios 46
6.8 Differences between applications :: $T(\mathrm{S}^A)$ [left] and $E(\mathrm{A}_S)$ [right] 47
6.9 WUP data - 16 runs of application *Borw8M1C* on system *DKTP* 48
6.10 Relative differences between versions (normalized to the mean of the specific combination) with *LPTP* 50
6.11 WUP data - Series of measurement runs with *Pressure* on *LPTP2* 51
6.12 PerfMon data - CPU Time per logical cores of *LPTP2* 52
6.13 Pressure results per workload step in absolute [left] and relative [right] values . . . 53

7.1 Representation of Experimental Setup II 54
7.2 Comparing %CPU Time values of different versions during a *DOCGEN* run 56
7.3 WUP data (in Watts) - Series of 6 measurement runs with *DOCGEN* 57
7.4 Overview of required time [left] and energy [right] per *DOCGEN* configuration . . 59

7.5 Overview of Base/App-distribution at energy [left] and power [right] values 59
7.6 Performance values per counter normalized to the mean over the specific counter . 61
7.7 %CPU Time per part/version of *DOCGEN* per CPU [left] and Process [right] . . . 65
7.8 Overview of Power per part/version of *DOCGEN* [left] and Ratio's [right] 65
7.9 Normalized performance counter values of *High Intensity*-parts per version 65

D.1 %CPU Time at *LPTP* per process during S^A [left] and S^B [right] 88
D.2 %CPU Time at *DKTP* per process during S^A [left] and S^B [right] 88
D.3 %CPU Time at *SRVR* per process during S^A [left] and S^B [right] 89
D.4 %CPU Time at *LPTP* per CPU during S^A [left] and S^B [right] 89
D.5 %CPU Time at *DKTP* per CPU during S^A [left] and S^B [right] 89
D.6 %CPU Time at *SRVR* per CPU during S^A [left] and S^B [right] 89

List of Tables

2.1 Research Questions linked to Activities 14

6.1 Overview of system specifications 37
6.2 Overview of application versions 38
6.3 Overview of the main measurement results per application per system 41
6.4 Overview of power during base state 43
6.5 Overview of determined $P(\mathrm{S}^M)$ and $P(\mathrm{S}^R)$ values (in Watts) 43
6.6 %CPU Time during measurements per CPU [left] and per Process [right] 45
6.7 Overview of determined $E(\mathrm{A}_S)$ [left] and $P(\mathrm{A}_S)$ [right] values 45
6.8 Ratios of $P(\mathrm{A}_S)$ to $P(\mathrm{S}^A)$ [left] and $P(\mathrm{S}^R)$ [middle], and %CPU Times [right] . . . 45
6.9 Differences between applications :: $T(\mathrm{S}^A)$ [left] and $E(\mathrm{A}_S)$ [right] 47
6.10 Overview of (average) non-CPU performance values at *DKTP* 48
6.11 Overview of average non-CPU performance values at *LPTP* and *DKTP* 49
6.12 Overview of workload at other components during measurements with *Systester* and *LPTP* 50
6.13 Overview of Pressure results per workload step in absolute and relative values . . . 52

7.1 Overview of the main measurement results with *DOCGEN* and *SRVR* 57
7.2 %CPU Time during *DOCGEN* measurements per CPU [left] and per Process [right] 58
7.3 The energy [left] and power [right] values of *DOCGEN* 59
7.4 Overview of power use range values and %CPU Times for *DOCGEN* 60
7.5 Overview of workload at other components during measurements with *DOCGEN* . 61
7.6 Overview of the main measurement results per part/version of *DOCGEN* 64
7.7 Overview of workload at other components per part/version of *DOCGEN* 64

D.1 Activity Table 90
D.2 Concept Table 91

Appendix A

Data Collector Sets

A.1 Experimentation Phase

The following two PerfMon data collectors sets were used during the *Systester* measurements.

Primary Set

Component	**Counter**	Description
Processor (per core)	%Idle Time	The percentage of time the processor is idle during the sample interval
	% Processor Time	The percentage of time that the processor spends to execute a non-Idle thread
Process (*Systester*)	% Processor Time	The percentage of elapsed time accounted to processing this process by the CPU(s)
	Private Bytes	The current size, in bytes, of memory that this process has allocated that cannot be shared with other processes
	Elapsed Time	The total elapsed time, in seconds, that this process has been running
	Thread Count	The number of threads currently active in this process, which is at least one thread.
Process (*other*)	% Processor Time	Incl. *System, Services, Idle*, & *Explorer*

Secondary Set

Component	**Counter**	Description
System	# Threads	The number of threads in the computer at the time of data collection
	# Processes	The number of processes in the computer at the time of data collection
	System Up Time	The elapsed time (sec.) that the computer has been running since it was last started
	System Calls/sec	Combined rate of calls to operating system service routines by all processes running on the computer
	Exception Dispatches /sec	The rate, in incidents per second, at which exceptions were dispatched by the system

Component	**Counter**	Description
System	File Read Bytes/sec	The overall rate at which bytes are read to satisfy file system read requests to all devices on the computer, including reads from the file system cache
	File Write Bytes/sec	The overall rate at which bytes are written to satisfy file system write requests to all devices on the computer, including writes to the file system cache
	File Control Bytes/sec	The overall rate at which bytes are transferred for all file system operations that are neither reads nor writes, including file system control requests & requests for information about device characteristics or status
	File Data Operations /sec	The combined rate of read and write operations on all logical disks on the computer
	File Read Operations /sec	The combined rate of file system read requests to all devices on the computer, including requests to read from the file system cache
	File Control Operations /sec	The combined rate of file system operations that are neither reads nor writes, such as file system control requests and requests for information about device characteristics or status
	Processor Queue Length	Number of threads in the processor queue
	% Registry Quota In Use	The (current) percentage of the Total Registry Quota Allowed that is currently being used by the system
	Context Switches/sec	The combined rate at which all processors on the computer are switched from one thread to another
Disk(s)	% Idle Time	The percentage of time during the sample interval that the disk was idle
	% Disk Time	The percentage of elapsed time that the selected disk drive was busy servicing read or write requests
	Disk Bytes/sec	The rate bytes are transferred to or from the disk during write or read operations
NIC	Bytes Total/sec	The rate at which bytes are sent and received over each network adapter, including framing characters
	Packets/sec	The rate at which packets are sent and received on the network interface
Memory	Available Bytes	The amount of physical memory, in bytes, immediately available for allocation to a process or for system use
	Pages/sec	The rate at which pages are read from or written to disk to resolve hard page faults
	Cache Bytes	The size, in bytes, of the portion of the system file cache which is currently resident and active in physical memory
	% Committed Bytes In Use	The ratio of “Memory//Committed Bytes” to the “Memory//Commit Limit”

A.2 Validation Phase

The following PerfMon data collectors sets were used during the *DOCGEN* measurements.

Overview Set

Component	**Counters**	**Details**
Process	% Processor Time	BriefServer, BriefConnector, System & Idle
		lssas, snmp, svchost#2, svchost#6, svchost#8
	Page Faults/sec	BriefServer, BriefConnector & snmp
	IO Data Operations/sec	
	IO Other Operations/sec	
Processor	% Processor Time	CPU0/CPU1 (per separate core)
	% Interrupt Time	
System	*See Systester "Secondary"*	Excluded: "% Registry Quota" & "System Up Time"
Disk		Excluded: "% Disk Time"
Memory		Excluded: "Pages/sec"
NIC		

Detailed Set per Component

- 6 component categories: Processor, Process, System, Disk, Memory, NIC
- We created a separate data collector for each component category
- Each collector includes all available counters per category
- **Location**: retrievable at the management server

Client Set

Component	**Counters**	**Details**
Process	% Processor Time	BriefInterface, BriefConnector, ServerApplet
	% Priv./User Time	System, Idle, services & explorer
Processor	% Idle/Processor Time	_Total (not per CPU core)
	% Priv./User Time	
System	*See Systester "Secondary"*	
Disk		Excluded: "% Disk Time"
Memory		
NIC		

Appendix B

Scripts

This appendix includes an overview of the script, that was used for automating the execution and monitoring of Systester. This script, named *PiBatch*, controls both the Management and Application systems. When the systems are prepared, the script is initiated from the Management system. Next, the arguments for the configuration of Systester can be filled in. The arguments enable the user to specify different arguments on the measurement level: (1) an additional “rest”-interval before the first series, (2) the number of series in the complete measurement by rebooting the application system each time (N.B. this was not possible with system *SRVR*), and (3) the amount of rest between runs as well as the number of runs per series. Then, on the application level, Systester also provides us the option to fill in arguments: (1) the number of Pi decimals to be calculated per run, (2), the number of concurrent threads (e.g., 2 threads entails processing 2 distinct calculations concurrently), (3) the CPU affinity (e.g. a specific single core or different combinations of multi-core processing), and (4) the calculating algorithm to be used. If the arguments are correctly filled in, the script will initiate Systester with the specified configuration. We added some additional feedback in the script to make it more-user friendly. However, we limited the amount of feedback that was given from the Application system to minimize the amount of interference. In addition, we also included an example script for using the *Pressure* application.

B.1 PiBatch - Management System

```
@echo off
:: Guide for setting CPU affinity
echo Overview Affinity (=Hexadecimal)
echo CPU3 CPU2 CPU1 CPU0  Bin  Hex
echo ———— ———— ———— ————  ———  ———
echo OFF  OFF  OFF  ON  = 0001 = 1
echo OFF  OFF  ON   OFF = 0010 = 2
echo OFF  OFF  ON   ON  = 0011 = 3
echo etc.
echo.

echo − Measurement Settings −
set /p delay=Set DelayTime (sec.):
set /p rest=Set Resttime (sec.):
set /p runs=Set Calc.−Runs:
echo.
echo − Systester Settings −
set /p decimals=Set Decimals:
set /p threads=Set Threads:
set /p aff=Set Affinity:
echo.

:: If System is SRVR: no reboot!
set /p reb=Set Reboots:
```

```
echo -- Algrotihm Part --
set /p alg=Gauss(1) or Borwein (2):

if ‘‘%alg%’’ == ‘‘1’’ set algo=gausslg else set algo=borwein
echo Selected Algorithm: %algo%
echo.

:: Initiate (Remote) PerfMon logging
echo --This procedure begins with starting MS PerfMon [takes some time]--
echo.

echo - Starting Perfmon @ %time% -
Call logman start tstsrv
echo.

echo --(Option) delay--
timeout %delay% /nobreak
echo.

:: Execute Reboot(s), e.g. Series
FOR /L %%i IN (1,1, %reb %) DO (

  echo --Wait %rest% Minutes before new Series--
  timeout %rest% /nobreak
  echo.

  :: Only if System is not SRVR!
  echo -- Initiating Reboot%%i --
  Start C:\Intel\PStools\PsExec \\*.*.*.* -u * -p * -d -i shutdown.exe /r /t 00

  :: Check current hour is single/double digit
  set HOUR=!time:~0,2!
  IF ‘‘!HOUR:~0,1!’’ == ” ” SET HOUR=0!HOUR:~1,1!
  set stamp=!date:~9,4!!date:~6,2!!date:~3,2!_!HOUR!!time:~3,2!!time:~6,2!

  echo --Initiating Measurement_!stamp! @ !time!--
  echo.

  :: Start PiBatch on Application system [Version Borwein or Gauss]
  Call C:\Intel\PStools\PsExec \\*.*.*.* -u * -p * -i ‘‘C:\Code\PiBatch\PIbatch_%
      algo%.bat’’ !stamp! %decimals% %threads% %aff% %runs% %rest% %%i
  echo.
)

echo --Cooling-Down (@ %time%)--
timeout 300 /nobreak

:: Stop PerfMon logging
echo - Stopping Perfmon @ %time% -
Call logman stop tstsrv
echo.

echo -- Completed Measurement_%stamp% --
echo.

:: Check current hour is single/double digit
set HOUR=!time:~0,2!
IF ‘‘!HOUR:~0,1!’’ == ‘‘ ’’ SET HOUR=0!HOUR:~1,1!
set stamp=!HOUR!!time:~3,2!!time:~6,2!

:: Update external log (e.g., an alarm when logged off from management system)
echo. Finished Measurement_%stamp% >> C:\...\...\temp\Endtime_%stamp%.txt

PAUSE
```

Listing B.1: Script on Management system

B.2 PiBatch - Application System

```
@echo off

:: Check current hour is single/double digit
set HOUR=%time:~0,2%
IF ‘‘%HOUR:~0,1%’’ == ‘‘ ’’ SET HOUR=0%HOUR:~1,1%

:: Check parameter(timestamp)
IF [%1]==[] (
  set stamp=%date:~9,4%%date:~6,2%%date:~3,2%_%HOUR%%time:~3,2%%time:~6,2%
) else (
  set stamp=%1
)

:: Check if local log file exists
if NOT Exist C:\Code\PiBatch\Logs\%stamp%.txt (
  echo.                                     >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo -- Log of Measurement_%stamp% --     >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo -- Created @ %time% --               >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo.                                     >> C:\Code\PiBatch\Logs\%stamp%.txt

  set /a new=1
) else (
  set /a new=0
)

:: Check parameters(Systester), if empty = local (test) initiation
IF [%2]==[] (
  echo - Systester settings-
  set /p res=Set Rest:
  set /p run=Set Runs:
  set /p dec=Set Decimals:
  set /p thr=Set Threads:
  set /p aff=Set Affinity:
  :: Number of Reboots
  set /a reb=0
) else (
  set dec=%2
  set thr=%3
  set aff=%4
  set run=%5
  set res=%6
  set reb=%7
)

:: Update local log
if %new%==1 (
  echo Algo: Borwein     >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo Rest: %res%       >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo Runs: %run%       >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo Decimals: %dec%   >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo #Threads: %thr%   >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo Affinity: %aff%   >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo.                  >> C:\Code\PiBatch\Logs\%stamp%.txt
)

:: Confirm Systester configuration
echo.
echo - Overview PI Input [%run% Run(S)] -
echo Rest (sec.): %res%
echo #Calc. Runs: %run%
echo Decimals: %dec%
echo #Threads: %thr%
echo Affinity: %aff%
echo.
```

```
:: Execute Run(s)
FOR /L %%i IN (1,1,%run%) DO (

  echo −− Taking some Rest before the Run−−
  timeout %res% /nobreak
  echo.

  echo − Run %reb%.%%i started @ !time!   >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo − Run %reb%.%%i started @ !time!

  :: Inititate Systester−cli with arguments
  [Borwein version] start ‘‘taart’’ /MIN /WAIT /NORMAL /AFFINITY %aff% ‘‘C:\Code\
      Systester\systester−cli’’ −qcborwein %dec% −threads %thr%

  [Gauss−Legendre version] start ‘‘taart’’ /MIN /WAIT /NORMAL /AFFINITY %aff% ‘‘C:\
      Code\Systester\systester−cli’’ −gausslg  %dec% −threads %thr%


  echo − Run %reb%.%%i finished @ !time!  >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo.                                   >> C:\Code\PiBatch\Logs\%stamp%.txt
  echo − Run %reb%.%%i finished @ !time!
  echo.
)

echo.
echo −− Cooling−down (@ !time!)
timeout %res% /nobreak
echo.

echo Measurement_%stamp% Finished @ !time!  >> C:\Code\PiBatch\Logs\%stamp%.txt
```

Listing B.2: Script on Application system

B.3 Pressure - CPU Energy Use Profile

```
@echo off

REM Command for starting Pressure Service
net start pressure

REM Command for setting CPU0 to 50 %
Call C:\Pressure\pressureconsole −x 0:50 1:1

timeout 100 /nobreak

REM Command for setting CPU0 to 100 %
Call C:\Pressure\pressureconsole −x 0:100 1:1

timeout 100 /nobreak

REM Command for setting CPU0 and CPU1 to 50 %
Call C:\Pressure\pressureconsole −x 0:50 1:50

timeout 100 /nobreak

REM Command for setting CPU0 and CPU1 to 100 %
Call C:\Pressure\pressureconsole −x 0:100 1:100

timeout 100 /nobreak

REM Command for stopping Pressure Service
net stop pressure
```

Listing B.3: Example script for Pressure

Appendix C

Structured Literature Review

- Date of search: January 16, 2014
- Digital Library: ScienceDirect (accessed through Utrecht University proxy)
- Science domains: Computer Science and Business, Management & Accounting
- Initial results: 262 (CS) & 32 (BMA) articles

C.1 Search Term String

("software architecture" OR "application architecture" OR "macro-architecture" OR "high-level design")

AND

(green AND (software OR it OR computing) OR (sustainability OR sustainable) AND (software OR ict))

AND

(metric OR measure OR evaluation OR assessment OR method OR analysis)

AND

(goal OR requirement OR driver OR decision OR concern OR principle OR aspect)

C.2 Remaining Articles

- Ardito, Luca & Morisio, Maurizio (2013). *Green IT - Available data and guidelines for reducing energy consumption in IT systems*
- Capra, Eugenio; Francalanci, Chiara & Slaughter, Sandra A (2012). *Is software "green"? Application development environments and energy efficiency in open source applications*
- Hoorn, Johan F; Farenhorst, Rik; Lago, Patricia & van Vliet, Hans (2011). *The lonesome architect*
- Kipp, Alexander; Jiang, Tao; Fugini, Mariagrazia & Salomie, Ioan (2012). *Layered Green Performance Indicators*
- Naumann, Stefan; Dick, Markus; Kern, Eva & Johann, Timo (2011). *The GREENSOFT Model: A reference model for green and sustainable software and its engineering*
- Zhang, He; Liu, Lin & Li, Tong (2011). *Designing IT systems according to environmental settings: A strategic analysis framework*

Appendix D

Figures & Tables

[add: introduction]

D.1 Workloads at Processes and CPUs during S^A and S^B

[add: description of section] (Nb. different CPU affinity at *LPTP*)

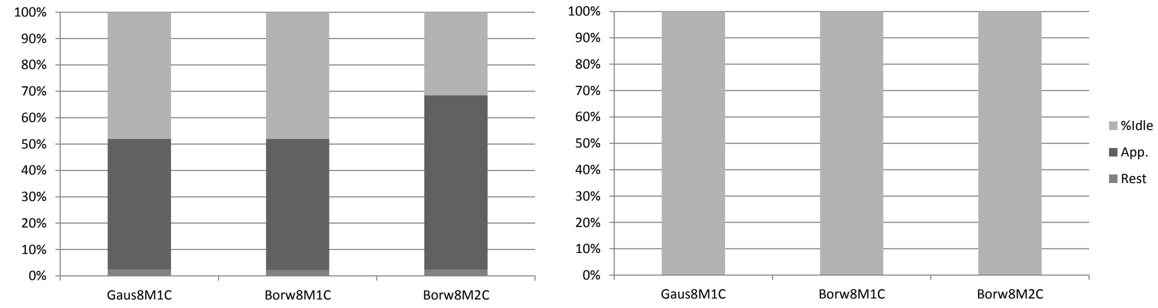


Figure D.1: %CPU Time at *LPTP* per process during S^A [left] and S^B [right]

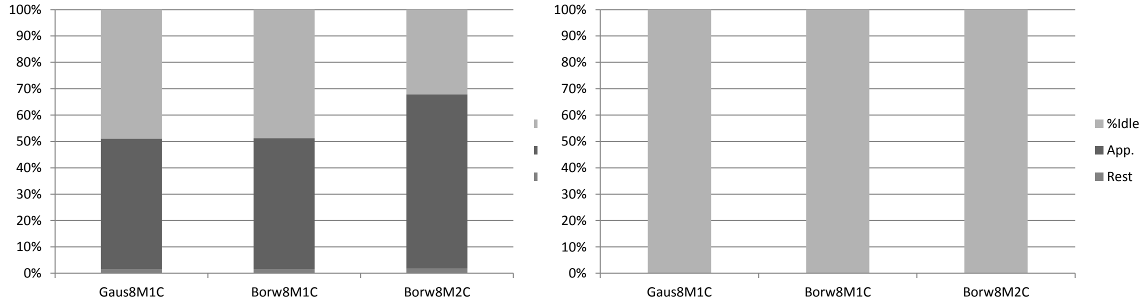


Figure D.2: %CPU Time at *DKTP* per process during S^A [left] and S^B [right]

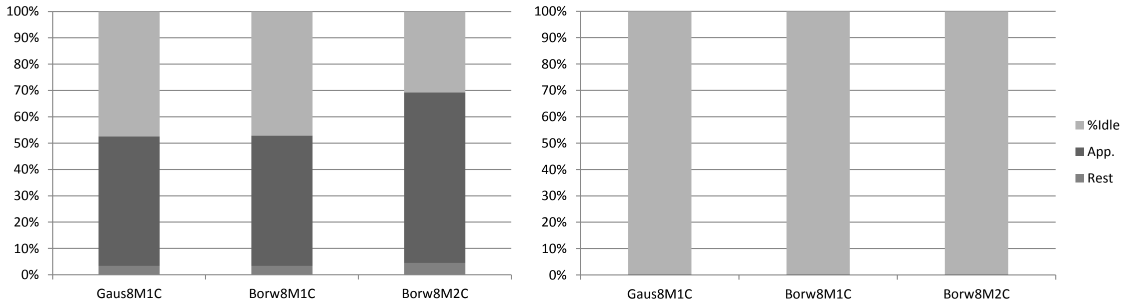


Figure D.3: %CPU Time at *SRVR* per process during S^A [left] and S^B [right]

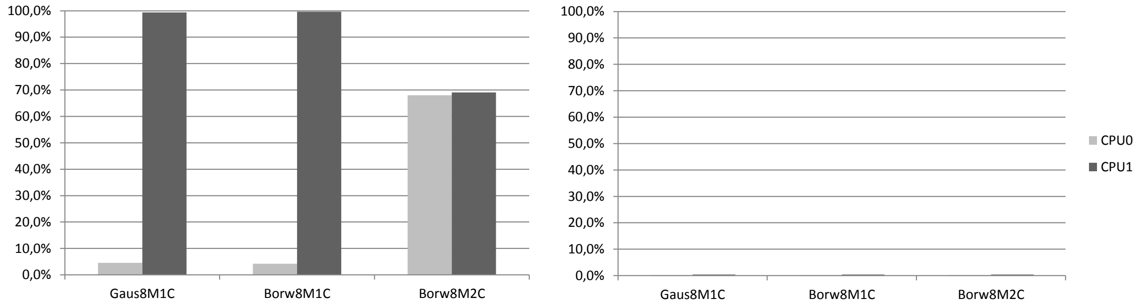


Figure D.4: %CPU Time at *LPTP* per CPU during S^A [left] and S^B [right]

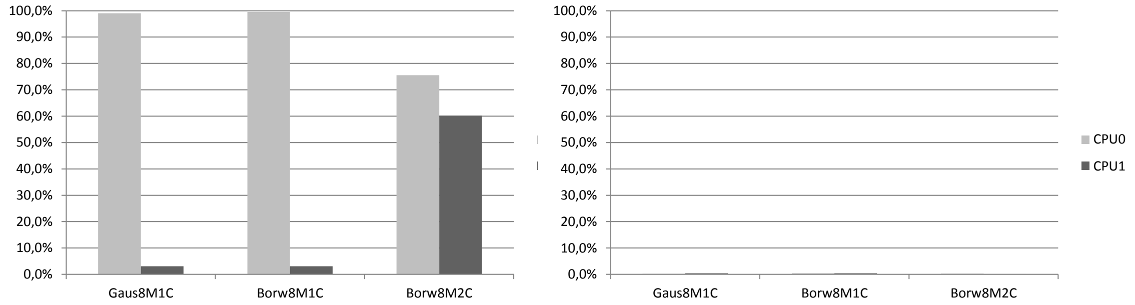


Figure D.5: %CPU Time at *DKTP* per CPU during S^A [left] and S^B [right]

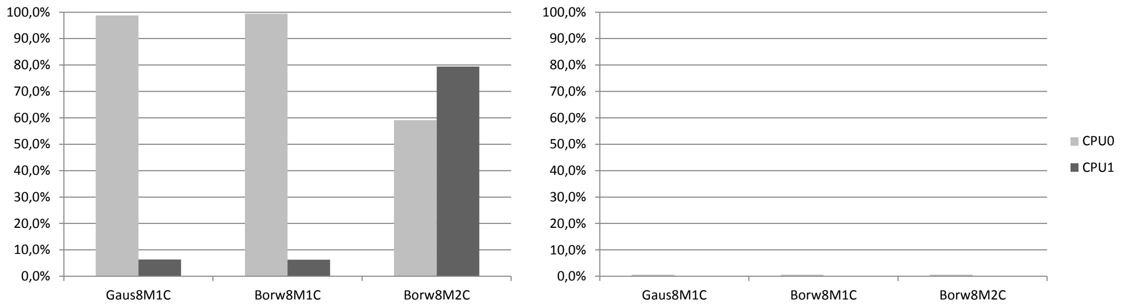


Figure D.6: %CPU Time at *SRVR* per CPU during S^A [left] and S^B [right]

D.2 Process-Deliverable Diagram

To complete the PDD constructed for this thesis (see Section 2.3) and provide additional descriptions, this appendix includes the respective Activity Table as well as the Concept Table.

Table D.1: Activity Table

Activity	**Sub-Activity**	**Description**
Literature & Company Study	Prepare literature study	A custom LITERATURE STUDY PROTOCOL is constructed after considering multiple SLR methods
	Search literature	LITERATURE is obtained by applying the determined search string at the set of selected digital libraries
	Process candidate literature	The LITERATURE COLLECTION is refined by applying a set of rules (e.g. inclusion/exclusion criteria)
	Search additional literature	Additional techniques are applied to increase the LITERATURE COLLECTION size (e.g. snowballing)
	Summarize literature collection	A LITERATURE SUMMARY contains main findings after analyzing the LITERATURE COLLECTION
	Create brainstorm protocol	A BRAINSTORM PROTOCOL is constructed to prepare the structure and ensure a certain quality
	Identify domain-expert(s)	The session requires multiple participants with relative work experience (e.g. a SA DOMAIN EXPERT)
	Conduct brainstorm session	Conducting the session results in a BRAINSTORM RECORDING, which can be analyzed afterwards
	Collect brainstorm session results	Multiple BRAINSTORM RESULT's are determined after analyzing the BRAINSTORM RECORDING
	Summarize brainstorm session	The BRAINSTORM SUMMARY contains main findings after analyzing the BRAINSTORM RESULT's
Development	Synthesize studies & build candidate metrics	After synthesizing the preceding studies, the KNOWLEDGE BASE and METRIC(s) are constructed
Experimentation	Develop experimental setup & protocol(s)	Both an extensive EXPERIMENTAL SETUP and EXPERIMENT PROTOCOL are constructed (and documented) to increase the quality of the measurements and support the reproduction of results
	Select & prepare an *AppSys*-configuration	Each specific APP/SYS CONFIGURATION needs to be prepared before a measurement is performed
	Perform measurement with configuration	Multiple MEASUREMENT DATA is created by a group of MEASUREMENT INSTRUMENTATION
	Process measurement data & analyze results	The Raw MEASUREMENT DATA is processed and combined in the MEASUREMENTS COLLECTION
	Write & submit paper on experiment results	Based on the analyzes of i.a. the EXPERIMENTATION RESULTS, a RESEARCH PAPER is written
Validation	Perform measurements with *DOCGEN*	By applying the VALIDATION PROTOCOL(s), additional measurements are performed and, also, the VALIDATION DATA COLLECTION is constructed
	Process measurement data & analyze results	The VALIDATION RESULTS are constructed by analyzing the VALIDATION DATA COLLECTION
Discussion	Reflect research results	(Including discussing the limitations of the research)
	Discuss research results	The meaning of RESEARCH RESULT is discussed (incl. its significance and expectations vs. outcome)
	Identify possibilities for future research	Different aspects of the RESEARCH RESULT, that can be applied in future research, are discussed
	Write thesis & present final results	Finally, all the main elements of the RESEARCH RESULT are combined in the THESIS DOCUMENT

Table D.2: Concept Table

Concept	**Description**
Literature Study Protocol	An approach for conducting a SLR based on the guide by [43]
Literature	A journal (article) or book
Literature Collection	A collection of LITERATURE instances
Literature Summary	A summary of the LITERATURE COLLECTION findings
Brainstorm Protocol	A structured approach for conducting a brainstorm session
Domain Expert	A person who is an authority in a particular (work) area or topic
Brainstorm Recording	An audio recording created during the BRAINSTORM session
Brainstorm Results	Definition of i.a. the problem, objectives or possible solution(s)
Brainstorm Summary	A documented summary of the BRAINSTORM RESULTS
Knowledge Base	The collection of findings that determine the experimentation
Metric (for software)	A (objective, reproducible and quantifiable) measure of a degree to which a software system or process possesses some property
Experiment Protocol	A structured approach for conducting a (set of) experiment(s)
Experimental Setup	The instrumentation, protocols, test systems and application(s)
Application Version	A specific version, or configuration, of the test application
System (Application)	The computer system on which the application is executed
Measurement Instrumentation	The hardware (incl. Management System) and software tools for measuring the energy consumption and performance of systems
App/Sys Configuration	Combination of an Application Version and Application System
Measurement Data	The raw data produced by the different INSTRUMENTATION
Measurements Collection	Processed, combined & analyzable MEASUREMENT DATA
Experimentation Results	The result of analyzing the MEASUREMENTS COLLECTION
Research Paper	A paper based on the Experimentation phase's main findings
Validation Protocol	A structured approach for conducting validation measurements
Validation Data Collection	Analyzable data that results from the validation measurements
Validation Results	Result of analyzing the VALIDATION DATA COLLECTION
Research Reflection	A collection of considerations and limitations for this research
Research Discussion	An overview of the meaning of the results (e.g. significance)
Future Research (options)	An overview of (created) opportunities for future research
Thesis (report)	An extensive document that describes the whole thesis project
Research Result	Incl. the main artifacts constructed during this thesis project