Release and Deployment at Planon: A Case Study

Slinger Jansen
Center for Mathematics and Computer Science
r.l.jansen@cwi.nl

Gerco Ballintijn
Center for Mathematics and Computer Science
g.ballintijn@cwi.nl

Sjaak Brinkkemper
Utrecht University
sjaak@cs.uu.nl

March 7, 2005

Abstract

This case study report describes the research results of a case study at Planon into the processes of development, release, and deployment. The research was done to document the release and deployment processes at Planon, to uncover strengths and weaknesses in these processes, and to compare Planon to other product software vendors. The case study was performed by doing interviews and examining development documentation, Planon software, and internally used tools. The results of the case study are organizational descriptions, Planon software descriptions, and the descriptions of the development, release, and deployment processes. The main conclusions of the case study are twofold. First the case study shows that striving for more integrated software knowledge management can relieve the processes of release and deployment. Secondly, the case study displays that extensive variation management can effectively increase the customer base for a software vendor.

1 Introduction

For vendors of product software it is becoming more and more difficult to manage and control the software configurations of all its users at the customer's site. It is labour intensive and error-prone to (semi)automatically register detailed lists of the software artefacts in use by each customer. To alleviate this problem the Deliver project proposes an Intelligent Software Knowledge Base that contains all facts about all artefacts together with their relevant attributes, relations and constraints. In this way, high-quality software configurations can be calculated automatically from a small set of key parameters. It also becomes possible to pose 'what-if' questions about necessary or future upgrades of a customer's configuration. This document describes a case study performed at Planon into the processes of release and deployment. The results of the case study are presented, existing of process descriptions of the development, release and deployment processes at Planon, a comparison to the Intelligent Software Knowledge Base, and an analysis of the results.

To study and define the scope of these problems in more detail we have performed a number of case studies at software developers and vendors to uncover the problems in the areas of software release and deployment. To find these problems we focus on the processes of development, release, and deployment, and the tools that support these processes. From these case studies we gather problems from the field and knowledge about tools used to support the afore mentioned processes [1]. The problem and process descriptions found during the case studies enable further research into the processes of development, release, and deployment of product software.

The contribution of this report is threefold. First, it describes the case study performed at Planon, an international software house that produces Facility Management and Real Estate management software for organisations. It describes the processes of development, release, and deployment for two product versions of Planons flagship product and the tools that support these processes. Secondly, this report displays the strengths and weaknesses of Planon in the area of release and deployment. Thirdly, this report displays what problems in the area of release and deployment have not yet been solved by Planon.

The report is built up as follows. Section 2 describes the aims of the Deliver research project in general, the aims of this case study, and describes the validity of the case study. Section 3 describes the organizational structure of Planon, its products, and gives a broad outline of the development processes at Planon. Sections 4 and 5 provide detailed descriptions of the development, release, and deployment processes for Planon 4 and Planon 5, from hereon referred to as P4 and P5. Finally, Section 6 discusses the processes Planon implements, the strengths and weaknesses of these processes, and the results of the case study.

2 Research Approach to the Case Study

This section describes the aims and validity of the case study at Planon. The case study, performed by the Deliver research group, attempts to uncover problems in the field of software release and deployment, by researching those fields at Planon. Planon is a rapidly expanding company that produces software to support facility management. This section first describes the conceptual model around which the case study is centered, and how it relates to the Deliver aims. The section continues by describing the aims of the case study and the actual research in more detail. To conclude this section describes the validity of the case study.

2.1 Deliver

The Deliver project studies the delivery, deployment and maintenance phases of software products. The research project is funded by NWO Jacquard and is based at the Centrum voor Wiskunde en Informatica (CWI). The focus of the project is as follows [2]:

"For vendors of product software it is becoming more and more difficult to manage and control the software configurations of all their users at the customer's site. It is labour intensive and error prone to (semi) automatically register detailed lists of the software artefacts in use by each customer. To alleviate this problem the Deliver group proposes an Intelligent Software Knowledge Base (ISKB) that contains all facts about all artefacts together with their relevant attributes, relations, and constraints. In this way, high quality software configurations can be calculated automatically from a small set of key parameters. It also becomes possible to pose 'what-if' questions about necessary or desired upgrades of a customer's configuration.

Managing software configurations is, however, only part of the story. They still have to be delivered to customers. To facilitate this delivery, the Deliver project studies how the computed difference between an existing configuration and a desired configuration can be used for the Web based Delivery of upgrades, furthermore, delivery protocols and implementation are studied. The results of the project are scientific publications, case studies and, in collaboration with industry, prototype tools."

The main aim of Deliver is to ease software release and deployment effort by managing software knowledge explicitly. The main areas of research for the Deliver project are configuration management, software products, and software deployment. The main focus of the Deliver group lies on large component based product software, such as ERP (Enterprise Resource Planning) systems. To manage this kind of software effectively, it is essential to carefully administrate the specific combination of standard components, tailored components, and customer specific components. The Deliver group offers an external assessment of release and deployment processes within the Software industry. These case studies supply the Deliver research project with knowledge about the problems in the industrial fields of release and deployment of software.

2.2 Definitions and Conceptual Model

Figure 1 displays part of the Deliver view of software delivery. At present software is delivered to the customer through some media (CD, floppy, Internet, etc) and then deployed at the customer site. The software vendors often do not have any knowledge about the customers' installed situation. Deliver sees the process of software delivery differently. Figure 1 shows the customer and vendor sharing knowledge about software artefacts by keeping a vendor and a customer knowledge base. The customer software knowledge base contains information on all deployed software artefacts at the customer site, whereas the vendor software knowledge base contains information on all deployable software available from that vendor. This knowledge base holds information about components, deployment restrictions, configuration details, etc.

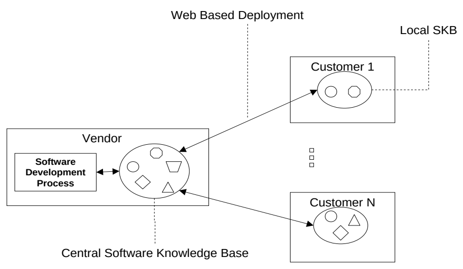

Figure 1: The Deliver View of Software Delivery

This knowledge can be used by the customer to deploy software, to ask ‘what-if’ questions, and to keep up to date with the most recent versions of software.

Two characteristics of this architecture improve the delivery process:

- The use of the Intelligent Software Knowledge Base (ISKB) that contains exhaustive information about all software artefacts and their constraints.
- A web based delivery process to deploy, upgrade, and replace software components based on the information in the ISKB.

By implementing the ISKB, many tasks that are part of the processes of release and deployment will be supported, so they can be done (semi-)automatically. Performing these tasks, such as consistent and complete deployment of components, will alleviate the workload on the software vendor and reduce the complexity of the release and deployment processes, thereby decreasing time-to-market. The release and deployment processes are defined as follows:

- **Release -** Hoek et al. [3] defines software release as “to package and make a software system available to a customer.” Looking at this definition, the process of releasing is not only the finishing step of development, but also packaging customer-specific installations.
- **Deployment -** According to Hall et al. [4], the deployment process potentially contains the “delivery, assembly, and maintenance of a particular software system at a site.” The deployment process description thus consists of detailed descriptions of the processes of delivery, assembly, and maintenance.

The processes described above have been used to map and direct the research because these processes are the prime focus of Deliver. Since all these processes are part of the software life cycle, the complete life cycle has been studied as well.

2.3 Intelligent Software Knowledge Base

The Intelligent Software Knowledge Base (ISKB) [2] stores exhaustive information on software artefacts and assists the software vendor in many ways during development. The focus of the Deliver group, however, lies in the areas of release and deployment. The ISKB implements features from product data management (PDM), software updaters [1], software deployment tools, and software configuration management. In this chapter the features implemented in our ISKB that are relevant to this case study are explained.

2.3.1 Structure

The ISKB supports and improves the processes of release and deployment. However, each of these processes poses different challenges for an implementation of an ISKB. Support for the release process with the ISKB is founded on the idea that the artefacts of every software system are stored in some kind of repository for development. The ISKB assists developers performing development tasks, such as product composition. These operations require knowledge about components and the relationships among them [5]. This software knowledge is stored in a versioned database and is accessible to all development personnel.

The knowledge stored in this development section of the ISKB can be published once a release is performed. This knowledge is publicly available for the ISKB to assist the process of delivery and deployment. The delivery process contains processes such as media creation, web delivery, and 'what-if' query handling. Finally, to assist and support the deployment of software at a customer site, a customer side knowledge base holds all the information on deployed software. This part of the ISKB assists the customer in finding out what components are required to get functionality not yet deployed on the customers' machine.

2.3.2 Usage

The following list summarizes some features and processes that are improved by implementing an ISKB as proposed by Deliver.

- **Version Control System (VCS) -** The Deliver project is not planning to implement yet another VCS; Deliver assumes, however, that products are stored in some kind of repository. This repository typically stores different versions of source code that can be extracted at will. Relevant to the Deliver group is whether dependencies are administered between different versions of sources in a VCS. This enables delivery of a component or software package with compatible sources, even if they are from different released software versions. The same holds for inter-component dependencies. The ISKB uses a feature description language (FDL) [6] that describes dependencies among components.

- **Build Systems -** The knowledge stored in the ISKB can be used to build products as well. When a product is designed, it is useful to use the dependency relationships between source files to assure consistency and completeness. The ISKB is able to provide dependency information to build systems or implements its own build system.

- **Customer Configuration and 'what-if' questions -** The ISKB implements mechanisms to generate an installation for a customer depending on the components the customer has installed. This means that some kind of querying mechanism is needed to generate a list (or tree) of what components need to be installed at the customer site to provide functionalities a customer has requested, or a list must be generated stating the conflicting components.

- **Customer Installation -** When the software is delivered to the customer site, a tool that deploys, configures and possibly builds the software is needed. Rollback functionality is a feature that is required for configurability. One of the main features Deliver wishes to see in the ISKB is support for web delivery. The ISKB should support all features mentioned here through some kind of web interface.

- **Software update and Installer Media Creation -** The ISKB should enable some kind of mechanism to do updates, and detect conflicting and inconsistent sets of components. There should also be some kind of auto update facility, so the applications based on the ISKB should support push, automatic pull, and pull mechanisms. The applications built around the ISKB should also support some kind of installer creation. This installation package can then be transported, through some media. It should be possible to create these installers automatically.

The list provided displays examples of process improvements and support by the implementation of an ISKB. The list does not only serve the purpose to inform, but also to compare to the solutions of Planon. The research will show that many of the concepts used by Planon are closely related or even similar to the concepts of the ISKB.

2.4 Research Questions

The main aim of our case study is to see the techniques applied by Planon and position them in our broader study of software release and deployment. To reach this aim we have the following research questions:

- **What do the processes of Release and Deployment look like for P4?** We want to see how P4 is developed and whether the approach to P4 is reused for P5. The techniques used for P4, though less advanced than those used for P5, supply a basis for studying P5 and the techniques used for release and delivery.
- **What do the processes of Release and Deployment look like for P5?** We want to see how P5 is developed and what the processes of release and deployment will look like upon release.
- **Can these processes be improved by using an ISKB or applying Deliver concepts?** Finally, we want to see if Deliver concepts can be applied to the release and deployment of P5.

We have attempted to get answers to these questions as precise and complete as possible. To do so a case study database and a case study protocol were created. To achieve correctness of the results we used the technique of cross-referencing our results. As a final check the work was critically reviewed by Planon. The case study results can widely be applied to other software vendors, because, even though Planon is active in one problem domain, its techniques are similar to software vendors in other domains. The case study protocol states that all information gained through interviews or document study should be double checked. Direct observations were critically reviewed before entrance into the database.

2.5 Research Methods

During the case study facts have been collected to answer the research questions. The means through which the Deliver group gathered these facts were:

- **Interviews -** The main research questions have been answered in part during the interviews with the people responsible for the development and usage of the Planon product.
- **Studying the software -** During our research the tools used to support development, release, and deployment were studied. Tools, such as the Release Wizard tool, were tested, evaluated, and documented.
- **Document study -** Planon provided us with guideline documents such as [7]. These documents were studied and added to our case study database.
- **Direct observations -** Direct Observations were made during our presence at Planon. These direct observations were later affirmed during the interviews.

3 Planon

Planon International is an international software vendor that produces Facility Management and Real Estate management software for organisations. Planon, founded in 1984, currently has a customer base of over 900 and has shown an annual growth of between 30 percent and 50 percent in employees over the last seven years. Planons products are marketed through four Planon subsidiaries and eight partners based in the Netherlands, Belgium, Germany, UK, Switzerland, and Austria. Planon made approximately 1.2 million profit with a revenue of 12 million in 2003. At present Planon employs 160 full time employees (see Figure 2).

Facility management is the discipline of ensuring functionality of the built environment by integrating people, place, process and technology[1]. The complex process of facility management encompasses activities such as long-range and annual facility planning, facility financial forecasting, real estate acquisition and/or disposal, work specifications, installation and space management, maintenance and operations management, and telecommunications integration, security, and administrative services. Planon develops client-server software (two- and three-tier architectures) with which it attempts to support the processes of facility management. Finally, Planon has implemented a set of components that can be used to communicate with other applications through XML interfaces.

Planon observes the following principles for its products. To begin with Planon uses the principle that common data can be used among different processes. The common data is therefore only entered once into

[1]International Facility Management Organisation - www.ifma.org

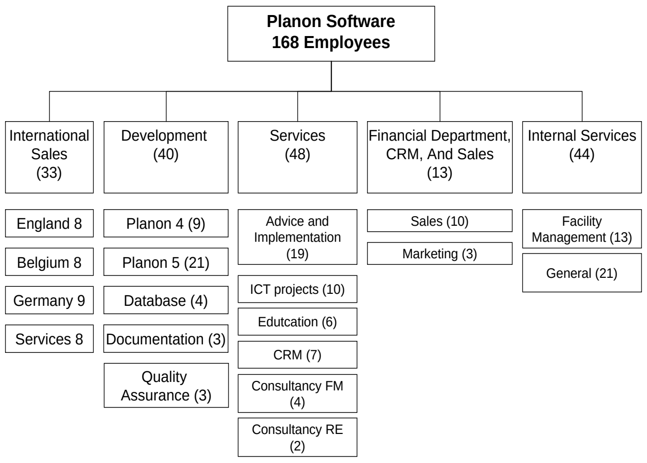

Figure 2: Planon Organisational Chart

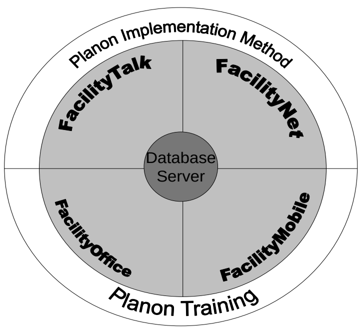

Figure 3: Facility Management Tools developed by Planon

the data model that is central to all Planon software. Secondly, because so many of the facility management processes are affected by Planon software products, Planon has developed its own implementation framework. With this framework a specific implementation path can be designed for a customer. Finally, Planon trains the application managers at customers on a regular basis, as to provide them with more competence with the Planon products.

3.1 Planon Facility Management Software Products

The separate processes that make up the combined activities of facility management are interdependent and consequently require an integral and coordinated management approach. According to Planon, significant efficiency improvements can only be achieved if facility management is supported by software that enables them to manage this integration and to respond faster and better to shifts in ‘customer’ requirements. To support in this need, Planon has created the Planon FacilitySolution. The Planon FacilitySolution consists of several product groups which together form a fully integrated facility management solution through a single central database. Planon FacilitySolution includes the following module groups (see Figure 3):

- **Planon FacilityOffice (FO) -** Planon FacilityOffice is aimed at supporting facility management personnel and management. The twentytwo Planon FacilityOffice modules have been designed to streamline and optimize internal facility management processes. This applies to both front and back office activities. Examples of the twentytwo modules are FO workorders, FO reservations, FO space management, and FO property management.

- **Planon FacilityNet (FN) -** With Planon FacilityNet it is possible to offer a wide range of facility management services to customers through an intranet or the internet. The FacilityNet modules are aimed at improving communication with the facility management customer and making the customer-oriented processes more efficient, by offering Planon functionality as a webservice.
- **Planon FacilityTalk (FT) -** Planon FacilityTalk is the generic solution for the creation of interfaces/links between Planon and other software systems/databases, such as financial, personnel and access control systems.
- **Planon FacilityMobile (FM) -** Planon FacilityMobile, also referred to as FacilityTools, include several specific programs which operate on mobile platforms, such as Palmtops and barcode registration equipment. These applications can exchange data with the Planon database through a small server built with the components from FacilityTalk.

The product that is described here, is P4. P4 has a two-tier architecture, consists of around 600,000 lines of code, is developed in Delphi, and works only under Windows. Planon has recently begun to develop P5, with a three-tier architecture, developed using J2EE. The first release of P5, planned for 2005, will contain a subset of the functionality of P4. Eight of the twenty two modules will be implemented by the time P5 is released. P4 and P5 use the same datamodel, to create interoperability during the transition period. The first release of P5 will be accompanied by the last release of P4. Planon software products are available in four languages. The software is delivered by a consultant that also performs the post installation tasks such as configuration, connection to other external systems, and creating the first data entries.

3.2 Planon Customers

Planon has 900 medium to large customers that deal with facility management on a large scale. Example customers are Achmea, ASML, Dutch Railways, KPN, Philips, and Shell. When a customer buys P4, the customer gets to choose the modules and the number of licenses for each module. Licenses are floating and user specific, i.e., a customer can have special roles for users of Planon software, of which a limited number can be logged in at the same time. Planon manages its customer relationships through the Planon Customer Information System (PCIS). PCIS stores, among the customer data, the modules and the number of licenses purchased by a customer.

3.3 Planon Employees

Planon has been growing continuously since its establishment in 1995. Currently Planon employs 168 people in four countries, as can be found in Figure 2. The organisational chart shows the division of Planon departments. There are three Planon locations, Zoetermeer, Nijmegen, and Wijchen, the Netherlands. The development department is in Wijchen, whereas the Zoetermeer location is a base for sales personnel to establish a larger customer base in the Randstad area of the Netherlands. The departments of interest to us are the development department and the services departments, because these departments are responsible for release and deployment of products.

3.4 Planon Research and Development

Planon development, as can be seen in Figure 2, is divided up into 5 teams. There are two Planon teams (4 + 5), a database team, a quality assurance team, and a documentation team. There are more people working on P5 at present, since that is the new version of Planon, whereas the others are performing maintenance and development tasks on P4. Interestingly, there is only one database team to support these two development teams. The reason for this is that the Planon products both need to work on the same database until all required functionalities of P4 have been implemented in P5. The quality assurance team checks the daily build version and strives to deliver a complete build each six weeks together with the documentation team, who provide the right documentation for a release.

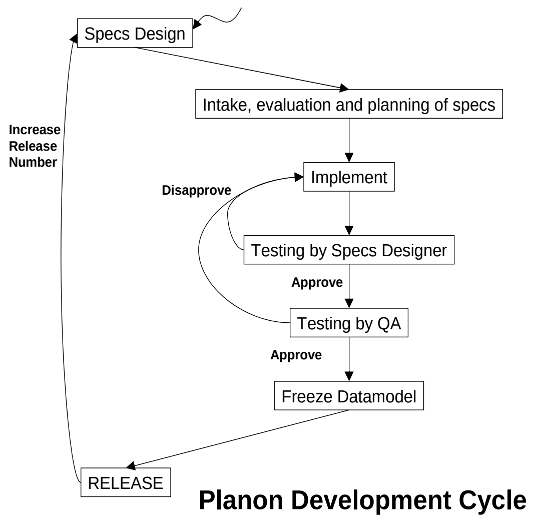

Figure 4: Planon Development Cycle

Planon develops all its products according to the cycle depicted in Figure 4. The picture shows that a designer starts by describing the requirements. These requirements are then implemented and tested by the designer. Quality assurance will then check and approve or disapprove the changes. Once approved by quality assurance, the product can be released. If the data model has changed for the new release the data model is frozen for that release. Once frozen, it can only be changed by increasing the matchcode of the data model (see Section 4.2). The cycle shown here is the cycle for a large release. For a smaller release, called builds within Planon, the database model is frozen only when necessary, and the build number is increased instead of the release number.

Planon uses Planon Facility Office to support its development team and two Facility Office environments have been implemented, being PCIS and DIS. DIS stands for Development Information System and is used during development of a new release only. The DIS is used by quality assurance, developers, and functional specs designers to supply them with information about bugs in the system. These groups also use CaliberRM, a requirements management tool, to evaluate, gather, and share information about requirements. None of the systems contain any workflow and all are only used to share information among the three groups that develop a product release.

The PCIS (Planon Customer Information System) environment is used throughout Planon. Development uses PCIS when bugs are reported which cannot be solved by customer support. The bug report is first evaluated by quality assurance to judge whether this is a design problem or a user problem. PCIS is used to store all information about and interaction with a customer. For more information on PCIS, see Section 4.3.1.

4 P4: Development, Release and Deployment

4.1 Development

This Section describes versioning, the software architecture, and the software structure of P4. P4 is developed using Delphi, a rapid application development system, based on the Pascal programming language. Planon uses Visual Source Safe for software configuration management. Much of the version structure used for P4 will also be used for P5 and shall be explained below.

4.1.1 Versioning

P4 is released yearly. After a release, a new branch is started with a higher release number. For example, when the new release 4.33 comes out, all new functionality has been added to 4.33 while all bug reports are solved on version 4.32 and if necessary on 4.33 as well. Besides the yearly release, P4 is built and stored in the release structure of Planon, as build numbers, e.g. 4.33.1.

While a new release is available, two of the previous major releases are still supported. There are three types of changes that are still being done on previous releases:

- **Bug fixes -** When a customer experiences a standstill, Planon will attempt to fix the bug in an older release. The reason for this is that the customer should not be forced to upgrade to a newer release.
- **Availability of new components -** If a new version of an important external component is released, and customers indicate they wish to use that external component, Planon will test compatibility of that component with the release used by that customer and, if necessary, changes are made to that version.
- **Small enhancements significantly increasing product quality -** In case an improvement is introduced in a newer release of P4 and it can easily be applied to an earlier release, Planon might choose to implement the improvement in an earlier release as well. This happens rarely, and usually at the request of a customer.

Finally, once a release is not as widely used anymore the release becomes unsupported. In our example in Figure 8, release 4.30 will soon no longer be supported. From then onward customers are required to update when they wish to get support. P4 has seen new releases on a yearly basis, where the actual release of 4.33 to customers, which is now in development, will be in 2005.

4.1.2 Software Architecture

P4 has been implemented using a two-tier architecture, displayed in Figure 5. The four components, FacilityOffice, FacilityMobile, FacilityTalk, and FaciltiyNet all access the database directly. FacilityNet is approached through a browser, whereas FacilityOffice represents the clients using the Planon executable. FacilityTalk is a separately running application that can accept incoming requests by (a number of times per minute) periodically reading all the files in a special folder with the use of XML interfaces. FacilityMobile is also a separately running server and receives connections from mobile appliances.

A clear downside of this architecture is that each tool addresses the database directly. If the datamodel changes even slightly, all clients and all servers (FacilityMobile, FacilityTalk, and FacilityNet) need to be reprogrammed with new interfaces. One of the main requirements for P5 is to solve this problem by building a three-tier architecture that decouples the database from the clients.

P4 is extensible through FacilityTalk. Other components can communicate with Planon products, for instance to create user accounts through Microsoft Windows domains instead of the FacilityOffice user manager. FacilityOffice is highly customisable and extensions are often required by customers. FacilityOffice can work with three different database engines, being Sybase, MS SQL, and Oracle.

4.1.3 Software Structure

A full installation of P4 consists of the following elements:

- **License file -** The license file is a coded file containing switches that activate modules in the Planon executable. The license file is unique for each customer and is generated from a contract stored in PCIS.
- **Manuals -** For a delivery of P4 there are manuals in each of the four language available.
- **Planon.exe -** The Planon executable contains all modules built for facility management by Planon. The executable is delivered to all customers.
- **Components**

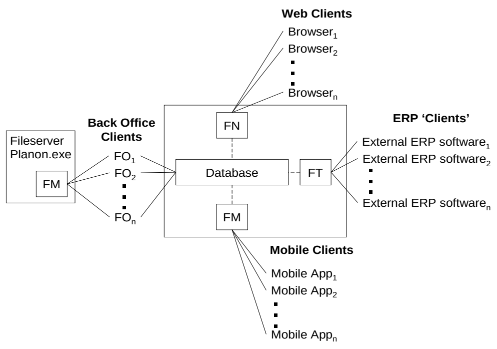

Figure 5: P4 software architecture

  - *External Tools* - External tools are components that are required for certain configurations, such as viewers like the Inso extension. External components are delivered to the customer through three channels. The component can be included in the Planon package, Planon functions as a reseller of the external component, or the customer can buy the product separately from Planon.
  - *Incorporated tools* - Incorporated tools are always included in the delivery, and are regularly included in Planon.exe.

- **Language File(s) -** Finally, a language file is included to provide customers of the system with the language they purchased.

The main variabilities are expressed by the license file, the external tools, and the language files. The license file states what external components have also been activated for a customer and what languages are available. At run-time the software will determine and set all the pointers to the right external components and pick the right language files. If external components have been purchased by the customer separately from Planon, the Planon deployment consultant can set the pointers to the right location for that component.

4.2 Release

Planon uses a versioned release structure to store all its deliverables. Once every six weeks the daily build of P4, which is used for testing, is stored in the release structure by quality assurance personnel. These builds get a number, such as 4.32.2 for the second build of version 4.32. Once these builds have been stored in the release structure they can be packaged into a product by the Release Wizard tool.

During development the data model is constantly changing. Such changes are recorded in conversion scripts. When changes are made to the data model that make it incompatible to previous versions of the software, the data model is stored with a new matchcode. A matching code for a build within a release and the data model means that they are compatible. For release 4.32 (see Figure 6), for instance, builds 10, 20, 21, and 22 have matchcode B. Builds 30, 40, and 41-43 have matchcode C. A customer using build 4.32.30 can update to build 4.32.43 without changing the data model, whereas a customer using build 4.32.22 cannot update to a higher build without updating the data model.

In June of 2004 Planon appointed a delivery manager to control released components and establish and test the relationships among them. The delivery manager has documented all compatibility relationships of external tools and components (FM, FO, FN, and FT). This knowledge can later be used to check whether a version of Planon can work with a version of an external component, such as AutoCAD.

4.2.1 Release Structure

Planon uses Visual Source Safe (VSS) as a configuration management system for source code. Besides source code, VSS also stores all available and published deliverables. These deliverables are assembled by the Release Wizard tool to create packages for customers. A portion of the total release structure can be seen in Figure 6 for P4.32. The Release structure stores all artefacts for each separate build of a release.

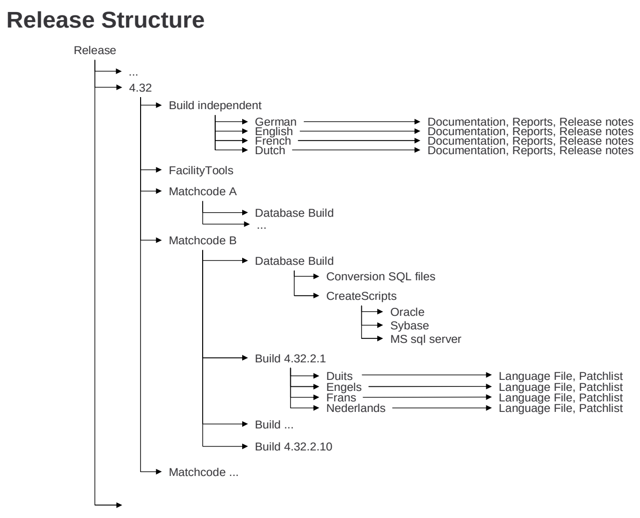

Figure 6: P4 release structure

To begin with, the build independent deliverables are stored, such as documentation, release notes, and reports. Secondly, the release structure stores all separate builds for release 4.32, which are categorised per database version. Each version of the database has a release number and a matchcode. Each of these matchcode categories contains a different datamodel and contains data conversion scripts to update from any previous database version to the current one. Builds that belong together with one version of a data model, cannot be used with other versions because the underlying datamodels are too different. The matchcode is compared on each connection of a client to the database, generating an error when the matchcodes are different.

4.2.2 Release Wizard tool

The Release Wizard tool creates a full deployment package from an .ini file, a license file, and the release structure in VSS. The license file specifies what languages are selected and what components have been activated for the customer. The Release Wizard tool also selects and includes the right language files. A full installation created by the Release Wizard tool consists of the parts described in 4.1.3. The Release Wizard tool then offers to burn these files onto a CD. In the Release Wizard tool is hardcoded what external components are required for which module, what modules are required for other modules, and what integrated tools must be delivered.

4.3 Deployment

The deployment of P4 at a customer is done by a consultant. The consultant first installs all external components. Then the consultant performs the installation and connects Planon software to other business information systems, such as human resources management systems using FacilityTalk. The delivery and deployment processes are shown in Figure 7. Figure 7 displays the complete deployment process. First, a contract in entered into PCIS. Then, the order bureau will generate a license file using the Planon Software Licenser. The license file and an accompanying .ini file tell the Release Wizard tool what components are required to create a package for this customer. Then, a consultant takes the package to the client and deploys it on a server with a setup utility. The server then contains a database, the Planon.exe, and optionally required components. The Planon.exe can be started by clients who can connect to a shared network drive on the server. Finally, Planon has also created a tool called Pclient, which puts a start-up link on the desktop to the shared drive and performs a number of checks on a client workspace.

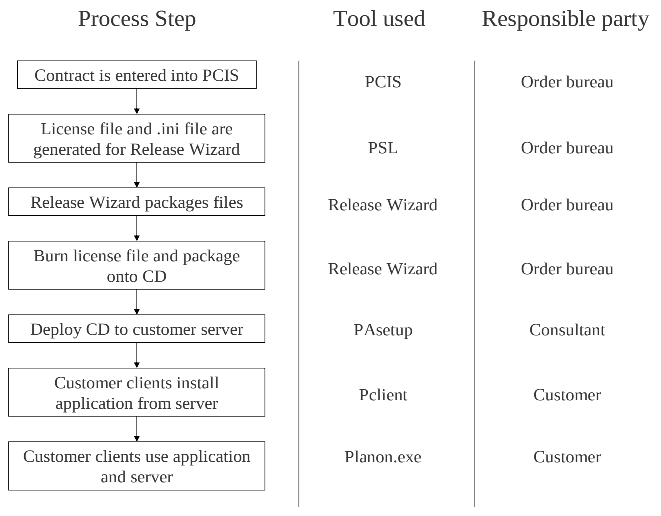
Process Step
Tool used
Responsible party
Contract is entered into PCIS
PCIS
Order bureau
License file and .ini file are generated for Release Wizard
PSL
Order bureau
Release Wizard packages files
Release Wizard
Order bureau
Burn license file and package onto CD
Release Wizard
Order bureau
Deploy CD to customer server
PAsetup
Consultant
Customer clients install application from server
Pclient
Customer
Customer clients use application and server
Planon.exe
Customer

Figure 7: P4 Delivery and deployment process and tools

4.3.1 Contracts and Deployment

PCIS is the Customer Relationship Management system (CRM). PCIS stores all information about customers, such as their components, their languages for the software, the customers support call data, and the customers used database version. All the information on customers is later used to derive the right license file for that customer, and thus the required components. The PCIS contains information on what modules are mutually exclusive, which is used by sales personnel when drawing up the contract.

4.3.2 Product Deployment and Configuration

To install and configure a Planon software product, the following tools are used.

PAsetup is the deployment tool used to install Planon onto a customer server. PAsetup copies all files to the server and stores the directories where deliverables are deployed. PAsetup does some checking of the environment and deploys external tools with their own installer scripts as well.

The **Pclient** is the program used by clients to deploy an activation link (shortcut) to the local workspace for Planon. The Pclient registers components and creates a shortcut to the Planon executable on the servers shared network drive. Pclient performs a number of checks before deployment occurs. The Borland Database Engine (BDE), which is supplied with Pclient, is only installed if it does not already exist in the customer configuration. If the BDE is already installed, the BDE version supplied with the Pclient is installed somewhere else than the standard location for BDE. The Pclient also looks for AutoCAD on the client platform for the use of some modules.

Planon delivers all modules together in a 11 Megabyte file named **Planon.exe**. This executable does not include external components such as viewers (see also Section 4.1.3 on the product structure of Planon). The executable is executed locally on the client machine, yet the Planon executable remains on the server site.

New fields can be added to each Planon screen. Such fields can be of any type, thereby enabling maximum flexibility. Such extra fields are already part of the database and do not present any update problems.

4.3.3 Updates and Upgrades

Since Planon knows what modules have been purchased by a customer, a Planon consultant who upgrades a configuration at any customer site knows exactly what external components, if any, must also be upgraded.

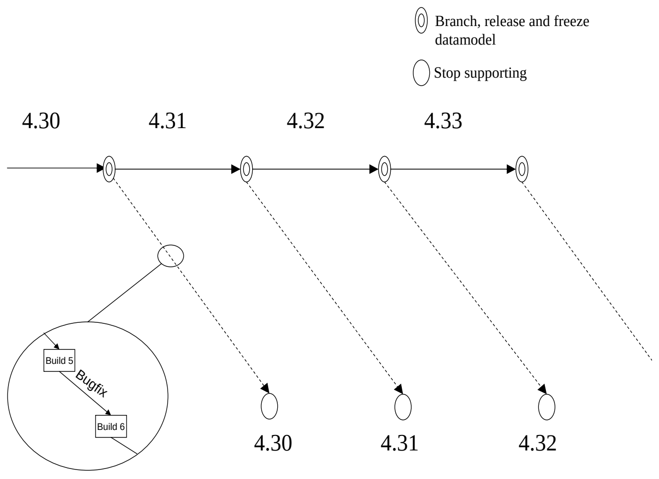

Figure 8: Planon version structure

When a new component is purchased by a customer a consultant needs to come by to reconfigure the system.

Within Planon a distinction is made between updates and upgrades. The dotted lines in Figure 8 indicate that bug fixes are still performed on the Planon executable of older releases. An update occurs when a customer goes from one build of a release to another, such as from 4.32.1 to 4.32.3. An “upgrade” means that a customer goes up to a newer release, such as from 4.30 to 4.33. Updates can be performed by customers by downloading the Planon executable and the database conversion scripts from an ftp site. Upgrades are usually performed manually with the assistance of a consultant. Also when a customer wishes to purchase new components, oftentimes a consultant must come by to reconfigure the software. The reason for the requirement of reconfiguration is that some external components might no longer work with a newer version of Planon. Since the entire Planon.exe file is replaced during an update, there are no internal dependency issues besides the database dependency. The database dependency is solved with the conversion scripts, which are only required if the matchcode for the database has changed.

4.3.4 Licensing

PSL stands for Planon Software Licenser and generates license files for customers and .ini files for the Release Wizard tool. The PSL downloads all customer information from the PCIS required to generate a license, such as the components that customer has purchased and the database version that customer will be using. Hard coded into the PSL are rules on what modules require other modules and what modules should be kept separate from each other.

5 P5: Development, Release and Deployment

P5 is a new product developed by Planon. P5 is planned to replace P4 within a few years and is built on a three-tier J2EE architecture. Planon has multiple reasons for introducing P5. To begin with Planon wants to integrate its four components, FO, FM, FN, and FT, into one package. Secondly, Planon wants to work with a three-tier instead of a two-tier architecture, to solve a number of version problems. If, for instance, the database changes slightly for P4, all module groups need adjusting, whereas P5 can circumvent this through the application server. Also, Planon wanted to reimplement many of the features, since they had been developed with differing requirements.

At the time of this case study, P5 is an unfinished and unreleased product. Many of the processes described for P4, such as updates, have not yet been implemented for P5.

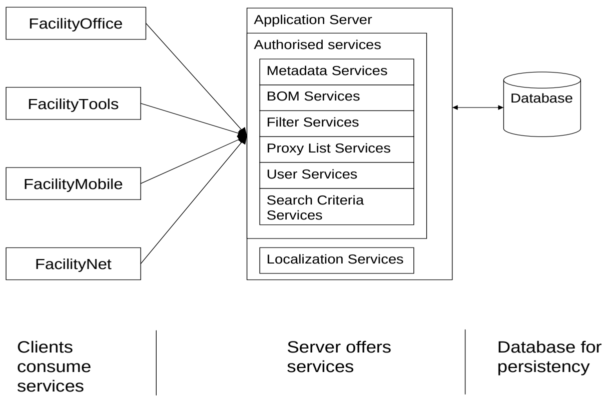

Figure 9: P5 software architecture

5.1 Development

P5 is developed using JBuilder and CVS. P5 is built on the J2EE platform. When Planon started designing the architecture for P5, they considered .Net and J2EE. They chose for J2EE because J2EE had been around longer and was well described and documented. At present there are no tools used to support developers besides CVS and JBuilder. The plan is to implement similar tools for P5 as have been implemented for P4, such as the Release Wizard tool and the PSL.

5.1.1 Software Architecture

The three-tier architecture of P5, as displayed in Figure 9, provides an application server, a database server, and clients. Clients connect to the application server to perform their operations. The application server handles these operations and connects to the database to store and provide the right data. Clients can be of any shape or size, because the application server can regulate and control all communication to the database.

For P4 the Planon executable contains all business logic. For P5 much of the business logic has been moved to the application server, such as authorisation services and database access routines. The advantages of moving this logic to the application server are threefold. First, the client is thinner (smaller) which makes it quicker to deploy. Secondly, the server can control requests and perform tasks such as load distribution over different servers. Finally, since clients do not approach the database directly anymore, many compatibility issues have been solved. For example, in case an older version of a client attempts to approach a newer version of the server, the server can handle that request safely. For P4 this would simply result in an error.

In P4 all four components were separately sold and delivered, whereas for P5 all components are delivered together. This approach has many advantages, such as the fact that there will no longer be version problems between these four components, assuming they are released consistently to begin with.

Extensibility for P5 is facilitated through FacilityTalk. The functionalities of FacilityTalk for P5, such as XML interfaces, are at present the same as for FacilityTalk in P4.

5.2 Release

Where VSS was the software configuration management system for P4, P5 uses CVS. Planon is planning to use CVS as the source code repository, but also as the configuration management system for deliverables. These deliverables can then be packaged by a tool similar to the Release Wizard tool. To simplify the process of release, Planon will store a full release with all components (internal and external) as a branch of a release tree, much like the release structure modelled in Figure 6. Because P5 has not yet seen a release, no release structure has been designed or implemented yet for P5. However, since the delivery of all components (including the database) will be delivered in one large EAR file, the release structure is expected to be less complex than the release structure for P4.

The Delivery Manager is at present not working on P5 yet, however, he is planned to be in charge of designing the release structure and use the knowledge from the P4 release process.

5.2.1 Software Structure

A P5 package will look as follows:

- **Planon Application Server classes -** A package containing all components of the Application Server, of which services are later activated and deactivated by the license file. The components providing these services are all stored in one large EAR file.
- **Planon client files -** The client files contain all files required for clients to run a client. The client files are deployed by Jboss or webstart. At present there is only one client that can be deployed in three different ways. However, future plans are to also have other clients, such as the clients running on mobile devices.
- **J2EE -** J2EE is included in each package of P5. J2EE is used to deploy the client software on user workspaces.
- **Manuals -** For a delivery of P5 there are manuals in each of the four languages available.
- **Language Files -** Language files are included in each package for internationalisation.
- **External tools -** External tools will also be included in a package of P5. Some of these external tools are stored in the collective EAR file, whereas others are supplied to the customer on a separate CD.

Similarly to P4, the variabilities are bound at package time and runtime. At package time the right external components are included in the package. At runtime the license file determines what components and languages are activated and deactivated. The only difference between P4 and P5 is that languages for P5 are all included in the delivery package instead of packaged separately by the Release Wizard tool, as is done for P4.

5.3 Deployment

Planon is planning to use the same methods for release and deployment as used for P4. The deployment of P5 is planned to follow the steps described in Figure 10. Similar to P4, a license file will still be generated from PCIS, Planons contract and CRM database. A future version of the Release Wizard tool will package the EAR file, language files, and external components from the concurrent versioning system.

A consultant can then deploy P5 at the customer site. The consultant will first install all external components and prerequisite applications, such as the database server and the J2EE appserver. Then the consultant will install all P5 components. Once installed, the consultant will start up the application server. During the installation a client directory is created on the server. The client can be used in three different ways:

- **Server side start-up -** Another option to start up a client is by going to the server and starting the client from the server, thereby storing the client into the RAM of the client computer.
- **Local installation -** The previous option causes a lot of network traffic per client start-up. As an alternative, Planon offers the option for clients to copy all files to a local harddisk. In this fashion the clients do not always have the most recent version, but it causes a lot less traffic with large numbers of clients.
- **Webstart -** Webstart starts the Planon client and checks for new versions from the application server. Webstart enables Planon to supply clients with the right version of the Java Runtime Environment (JRE) and the client software. Webstart has the advantage that it works on different platforms, however, due to some recent problems with different versions of the JRE Planon is considering to drop the use of Webstart altogether.

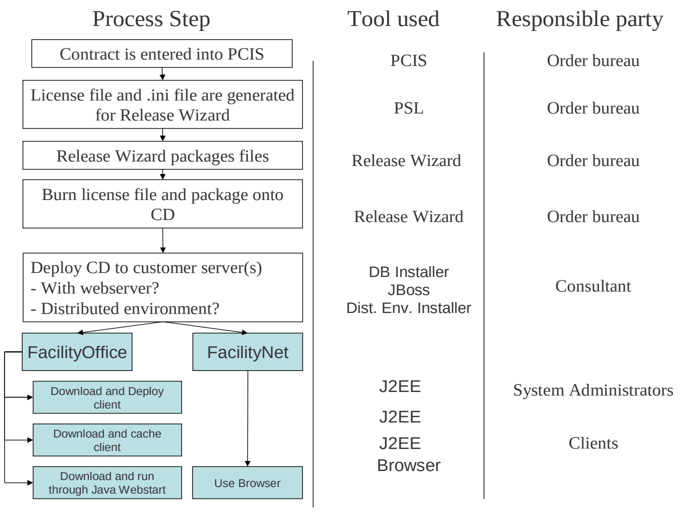
Process Step
Tool used
Responsible party
Contract is entered into PCIS
PCIS
Order bureau
License file and .ini file are generated for Release Wizard
PSL
Order bureau
Release Wizard packages files
Release Wizard
Order bureau
Burn license file and package onto CD
Release Wizard
Order bureau
Deploy CD to customer server(s)
- With webserver?
- Distributed environment?
DB Installer
JBoss
Dist. Env. Installer
Consultant
FacilityOffice
FacilityNet
Download and Deploy client
Download and cache client
Download and run through Java Webstart
Use Browser
J2EE
J2EE
J2EE
Browser
System Administrators
Clients

Figure 10: P5 release and deployment process and tools

5.3.1 Product Deployment and Configuration

One of the requirements for P5 is that it should be highly configurable to accomodate different problem domains. To fulfil this requirement, Planon uses configurable user application forms, which can be connected by a navigation structure. These configurable user views allow for different domain-specific solutions, using the same datamodel, by defining new configurable user views and navigation structures. Planon is planning to sell these sets of navigational structures and views as domain specific solutions. Customers can also purchase an application that builds these navigational structures and views so they can fully configure P5 themselves.

Configuration data is mainly stored in the database. Examples of such data are licensing information, user specific data (language preferences, passwords, etc), the configurable user views, and navigational structure. User data, such as language preferences and screen settings, is stored centrally in the database to enable users to log in from any client. One exception to the configuration data are reports. Planon is planning to use a third party report generator, which will access the application server to generate these reports. They will be stored in some central location, such as the application server, to enable sharing.

P5 supports four languages, but no support is specifically built in for internationalisation besides currencies and date formats. Currently the plan is to handle such variability with run-time variation mechanisms. Should such mechanisms be insufficient, then Planon can perform customisation work on the source code and deliver different releases to different countries. The last option is a worst case scenario for Planon and Planon expects that run-time variation mechanisms and the configurable navigation structures and database views are sufficient.

5.3.2 Updates and Upgrades

P5 will be upgraded by overwriting the previous version with the newest version of P5. This is the same as P4, where they overwrite the Planon executable. In this fashion the database will remain a separate component with its own version number. The database still needs to be updated through database update scripts.

6 Discussion

In the discussion of the Planon case study we will first compare Planons solutions to the ISKB techniques (see Section 2.3). We will indicate the strengths and weaknesses of Planon and its development, release, and deployment processes. Finally, we will show how Planon could, with the introduction of ISKB techniques, use fewer resources and increase product quality.

6.1 Planon Strengths

Planon uses many techniques that are similar to the ISKB techniques. The release structure that is used to store deliverables, which is stored in a configuration management system, is much along the lines of an ISKB. The Release Wizard tool that packages these stored deliverables according to predefined logic also ties in well with an ISKB. We also believe that integrating the CRM system and contract management system into the delivery process enables Planon to deliver software with fewer resources.

Planons' strengths can be found in the following:

- **KISS -** As seen in our previous work [8] the KISS (Keep It Short and Simple) perspective can be effective in minimizing effort in the software development cycle. One application of the KISS perspective by Planon can be found in the fact that they deliver the complete application and activate the purchased functionality. Another example is the fact that the data model is seen as one component for different products, which reduces cost of development. This generalization makes it possible for different pieces of software to use the database. Another example of KISS is that all (user) configuration data is stored in the database. This is a lot less complex than storing this information locally, or on the application server in some file format.

- **"Eat your own dogfood" -** Planon, as well as for instance Microsoft [9] and Exact Software [8], uses its own products internally. Planon uses its facility management tool internally for the management of its internal planning, for facility management, for customer relationship management and to distribute developer information among development teams. The internal use of these commercial tools results in quick internal feedback.

- **Deliverables stored in SCM System -** Planon stores its deliverables in the SCM system it is using. These deliverables are versioned and enable Planon to review old releases and fix bugs in them.

Many of these strong points were also observed at other companies in the product software sector.

6.2 Weaknesses and Opportunities

With respect to the areas of release and deployment Planon can still improve many of its processes. Different factors account for these weaknesses, which leave opportunities for improvement for Planon. We found weaknesses in the areas of knowledge management and resource efficiency.

6.2.1 Knowledge Management

In the area of knowledge management Planon is trying to improve the processes of release and delivery. Generally, Planon has found that knowledge about the software is too spread out. Knowledge about the released and deployed software at Planon is present in human readable format and in application readable format. To begin with, the application readable format is present in the release structure containing all the required artefacts for each release, including database conversion scripts. Secondly, the Release Wizard tool contains hardcoded information on what components are required to create a complete installation and where to find these components in the release structure. Finally, the Planon Software Licenser can derive a license from the license information stored in PCIS and contains knowledge on dependencies between modules.

The human readable format is present in PCIS, which contains knowledge on what components are mutually exclusive, and what components explicitly require each other. Secondly, the delivery manager at Planon has created a document that contains all supported compatibilities between external components and versions of Planon. Finally, the service department knows all about the procedures and problems related to performing an update.

- **Relationships too complex -** The efforts of the delivery manager to manually document component compatibility relationships have been fruitful, but will prove to be too complex and too extensive in the future. The number of components and the relationships among them are already too numerous for P4 to be managed manually. We would like to propose the use of a PDM system to automate and solve the problems stated.

- **Software development process knowledge too spread out -** Product and release planning information is currently only in the heads of the Planon development department. Such information should be centrally stored to inform all Planon employees of the status of a product and its state of development [7].
- **Development tools not managed explicitly -** Planon has developed many tools internally, and is also using configured versions of external products to support development. These internally used (development) tools are not managed explicitly. These tools should be seen as part of development and should be managed in such a way that these tools are available to the people who need them, in different versions. The same holds for responsibility for these tools, since often it is unclear who is responsible for a certain development tool and who is not.

The problems stated above offer Planon opportunities for improvement. Clearly, the knowledge about released software at Planon is too spread out. Much of the human readable information should be present in a centrally stored location, such as a website, and the application readable information is spread out among different applications. The latter is a more serious problem because these tools are not explicitly managed, and therefore the maintenance of the tools cannot be guaranteed.

6.2.2 Resource Efficiency

Planon can also improve its release and deployment processes to improve resource efficiency. Planon has the following weaknesses in the area of resource efficency:

- **Components can be packaged together without being compatible or complete -** The release wizard does not enforce correctness criteria when releasing a product, thus allowing the releasing of a product which is incomplete or contains components that are incompatible. Such correctness criteria should be centrally available to different applications.
- **Workflow is not integrated with the SCM system -** By integrating information from a workflow system with an SCM system development becomes traceable. Such traceability makes the software development process more reliable since each change can be traced back to a bug or request for change.
- **Customers get manual procedures for updates -** Customers get procedures for downloading and deploying updates onto its configuration. Such steps are frequently cumbersome and error prone.
- **Information about a Planon configuration can only be sent to Planon manually -** Planon support uses a tool called Planon Inspector to gather information about a Planon configuration by gathering all the files, file versions, and configuration options of Planon FacilityOffice in Windows. The Planon Inspector report can then be sent to Planon by e-mail.

Each of the weaknesses stated above provide an opportunity for improvement at Planon. To begin with correctness criteria can be enforced by centrally managing product compatibilities in a computer readable format. Secondly, workflow and SCM can be coupled by attaching a label in the SCM when a workflow step has been performed. Thirdly, a framework can be implemented to support the complete update process. Finally, the Planon Inspector should be an integrated part of the software for Planon 5, thus allowing automatic feedback upon a failure, when the client is connected to the internet.

6.3 Evaluation and Opportunities

Planon is a strong tool builder, and has developed many tools for the release and delivery of software, such as the Software Licenser, the Release Wizard tool, and the Planon Facility Office DIS environment. However, Planon would have preferred using external tools for these processes, had they been available, since that could have reduced cost of development. Planon especially expresses the need for a universal release and delivery framework, supported by various tools [1].

Planon could do many things to improve the processes of release and delivery. To begin with knowledge about software products and tools should be available centrally, such as release notes and compatibility information. We would like to propose to Planon the use of a development tool such as Maven[2], which, alongside development support, can publish knowledge about the software on a weblocation. We would also like to propose to Planon some form of a PDM tool, where workflow can be attached to deliverables, such that work will be traceable. Finally we would like to propose an application that can assist in publishing and using information about component compatibilities. The current solution where compatibility information is gathered manually is insufficient for Planon, because the number of components is always growing, with an exponential number of dependencies.

Planon has chosen to start developing P5 to use state-of-the-art technology for development to enlarge its market share. Many choices Planon has made for the implementation of P5 will largely increase the number of problem (sub)domains in which the software can operate. As an example, the usage of configurable user screens and navigation structures will allow Planon to implement its software in more (sub)domains. The use of these user screens and navigation structures require smart knowledge management, however, in the form of a PDM system. The PDM system can support processes such as distribution and evolution of the screens and navigation structures. These preconfigured domain specific solutions can assist in implementing Planon quicker at a customer.

Planon is making some restricting choices when it comes to delivery of software. At present all software is delivered to customers through a consultant. This makes the delivery process labour intensive. This is no problem as long as Planon keeps delivering large implementations of its software to large customers that require a lot of extra configuration options. However, if Planon wishes to deliver a “lite” version of its software they need to largely improve the deployment process. The ISKB already suggests some kind of webdelivery to customers, and in the case of Planon a web delivery could save the deliverer time, and saves the cost of burning a new CD each time an installation is created.

At present Planon uses conventional channels, such as mail, customer days, and e-mail, to reach its customers and inform them of new developments of Planon software. The same information, however, could be spread by the application, if it connected to the internet regularly. Such information should of course only be seen by the application manager(s). Once introduced to Planon, they coined the facility of providing information through the application as the ”Planon Customer Web”.

6.4 Conclusions and Outlook

We are planning to further define the problems found in the case studies [8], and build prototypes to support the release and deployment processes. Such prototypes will enable distribution of software configurations in distributed environments, enable filling of release structures and PDM tools, and support the deployment of software on client sites.

Previously we have undertaken similar research at Exact Software [8] to uncover problems in the areas of release and deployment. Other work, performed by Tiihonen et al [10], describes the state of the practice of configuration management and the deployment process in the software industry of Finland. Jaring and Bosch [11] have studied variability of MRI scanner software and is different from our work in that our work focuses on enterprise resource planning software.

Planon is a successful software company in the area of facility management, considering they are currently the leading producer of facility management software in the Netherlands. Planon implements many of the ISKB concepts. Planon stores and manages all released versions, Planon stores deployment information about each customer, and Planon has implemented its licensing mechanisms to be part of the delivery process. Many of these implementations have already lead to more cost efficient delivery and deployment of software. However, Planon can effectively decrease costs and improve product quality by implementing some other concepts of the ISKB. Planon can decrease its cost by explicitly managing its internal tools, by centrally storing software knowledge, and by automating more parts of the release and deployment processes. Planon can increase its product quality by improving its software artefact management and by automating the deployment process to include web delivery. Planon could also profit from the implementation of the Planon Customer Web and explicit management of user configurable environments in a PDM.

[2]maven.apache.org

This report describes the observations done by Deliver during a case study at Planon. The final conclusions are that Planon did not yet implement all the concepts of an Intelligent Software Knowledge Base and can reduce costs and increase product quality by implementing more of these concepts.

References

[1] S. Jansen, S. Brinkkemper, and G. Ballintijn, “A process framework and typology for software product updaters,” in *European Conference on Software Maintenance and Reuse (CSMR ’05)*. IEEE, 2005.

[2] S. Brinkkemper and P. Klint, “Intelligent software knowledge management and delivery,” 2003.

[3] A. van der Hoek, R. S. Hall, D. Heimbigner, and A. L. Wolf, “Software release management,” in *Proceedings of the Sixth European Software Engineering Conference (ESEC/FSE 97)*, M. Jazayeri and H. Schauer, Eds. Springer–Verlag, 1997, pp. 159–175.

[4] R. S. Hall, D. Heimbigner, and A. L. Wolf, “A cooperative approach to support software deployment using the software dock,” in *International Conference on Software Engineering*, 1999, pp. 174–183.

[5] T. van der Storm, “Variability and component composition,” in *ICSM*, 2004.

[6] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, “Feature-oriented domain analysis (FODA) feasibility study,” SEI, CMU, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.

[7] R. Traets, “Deliver management: Inventarisatie van wensen, ideeen en behoeften.” Planon, 2003.

[8] S. Jansen, G. Ballintijn, and S. Brinkkemper, “Software release and deployment at exact: a case study report.” CWI technical report.

[9] M. A. Cusumano and R. W. Selby, “Microsoft secrets.” Free Press, 1995.

[10] J. Tiihonen, T. Soininen, T. Mannisto, and R. Sulonen, “State of the practice in product configuration – a survey of 10 cases in the finnish industry,” in *Knowledge Intensive CAD, First Edition*. Chapman et Hall.

[11] M. Jaring, R. L. Krikhaar, and J. Bosch, “Representing variability in a family of mri scanners,” *Softw. Pract. Exper.*, vol. 34, no. 1, pp. 69–100, 2004.

Acknowledgements

We would like to thank Planon for enabling us to do the case study and present our results. We would like to especially thank David Griffioen, Erik Jaspers, Bertin Kiekebosch, Wim Heutz, and all the other interviewees. Finally, we would like to thank Tijs van der Storm for his helpful reviews and discussions.