A Case Study of the Release Management of a Health-care Information System

Gerco Ballintijn
Centrum voor Wiskunde en Informatica (CWI)
Amsterdam, The Netherlands
g.ballintijn@cwi.nl

Abstract

This paper describes a case study of the release management of CS-ECIS, a health-care information system developed by the Dutch software vendor ChipSoft. We performed this case study to gain insight into the release management activities of a real-life company. This insight would enable us to propose new ways to ease the effort and reduce the risks associated with release management. The case study consisted of recording both the release management activities for CS-ECIS and its related development and deployment activities, and subsequently comparing these activities with our initial release management model. The description and comparison enabled us to both evaluate and improve this initial model.

1. Introduction

Release management of enterprise application software is a complex task for a software vendor. This complexity is caused by the enormous scale of the undertaking. There are many customers to serve, which all might require their own version of the application. Furthermore, these applications frequently consist of many components that depend on each other. On top of that, the components evolve over time to answer the changing needs of customers. Consequently, releasing these applications takes a significant amount of effort and is frequently error-prone.

The goal of our research project, called Deliver, is to ease the software release management effort and reduce its risks by managing explicitly all knowledge about the software. For instance, by managing the software knowledge explicitly, a vendor can improve its software update process by enforcing consistency requirements. Furthermore, the explicit management of software knowledge enables the evaluation of "what if" scenarios, such as what will happen to the software configuration of a customer, when she updates a certain component?

To ease the effort and reduce the overhead of software delivery, we also desire support for software delivery via the Internet, both as full packages and as incremental updates [14]. Fortunately, the explicitly managed software knowledge can also be used to enhance the delivery of software. Specifically, it can be used to automatically create incremental updates by computing the difference between an existing software configuration and a desired configuration.

Central to the release management activities in our model is the Intelligent Software Knowledge Base (ISKB). This knowledge base can be seen as a type of software product data management system (PDM), in that it stores information about all the artifacts that are part of an application's life cycle. To design this knowledge base, we need insight into the real-life issues of releasing large scale enterprise applications and about the context in which these issues occur, such as development and deployment activities.

After having performed case studies at the Dutch software vendors Exact Software [12] and Planon [13], we decided to examine release management at the Dutch software vendor ChipSoft. ChipSoft is a successful medium-size company that develops health-care information systems. The case study had two goals. The first goal was to describe the development, release, and deployment activities of ChipSoft, including the tools and techniques used to support them. The second goal was to compare these activities to the initial model we developed in the Deliver project. The main contribution of this paper is the resulting description and comparison. A technical report [2] describes the case study in more detail.

The rest of this paper is structured as follows. Section 2 provides an overview of the Deliver model. Section 3 describes the design of our case study and Section 4 gives the result of the case study: a description of ChipSoft, the products it sells, and its development, release, and deployment processes. In Section 5, we evaluate our findings with respect to the Deliver model. Section 6 discusses related work, and in Section 7 we draw our conclusions.

2. The Deliver Model

According to SWEBOK [20][1] *software release management* "...encompasses the identification, packaging, and delivery of the elements of a product, for example, executable program, documentation, release notes, and configuration data." In this paper, we use a slight variation on this definition, where we put the "making available" activities in the release management phase and the delivery activities in the *software deployment* phase. Furthermore, we consider *software development* to be all activities before release management, such as design and implementation, and we consider software deployment to be all activities after release management, such as distribution and configuration.

The release management of a software application is complicated by several factors. The first factor is the number of components the application consists of. The second factor is the number of supported versions (i.e., revisions and variants) of the components. The third factor is the number of other applications the components are shared with. Consequently, release management is particularly difficult when applying structured reuse approaches, such as software product families, product lines, or product populations [23].

In the management model proposed by the Deliver project, the Intelligent Software Knowledge Base (ISKB) plays a central role throughout the software life cycle. It can be used to support development, release, and deployment activities. Furthermore, the ISKB can be used both to drive development, release, and deployment process and to verify product quality and progress. The ISKB also simplifies the management of these processes by gathering and combining all relevant information. In earlier work [17], Meyer also describes an ISKB, but this one is focused primarily on software development.

Conceptually, the ISKB can be considered a type of software product data management (PDM) system [8], in that it stores exhaustive information on potentially all software artifacts. These artifacts can be both original and derived, and both available at the vendor and installed at its customers. The information has to be exhaustive since the release management activities tightly integrate with both development and deployment activities, and all three require different kinds of information. Apart from information usually stored in a PDM, we expect the ISKB to also store information that can be found in version management systems and customer relationship management (CRM) systems.

The ISKB consists of two parts. The first part is the Central Software Knowledge Base (CSKB), which is stored at the vendor site. This knowledge base stores information about all the artifacts needed for all *available* releases of all available products. The second part is the Local Software Knowledge Base (LSKB), which is stored at the customer site. This knowledge base stores information about the products that are actually *installed* at the customer site (see Figure 1 and the description below). The LSKB is similar in functionality to the local RPM database of the RPM package manager [1] and the on-line dependency analyzer (OA) developed by Microsoft Research [7].

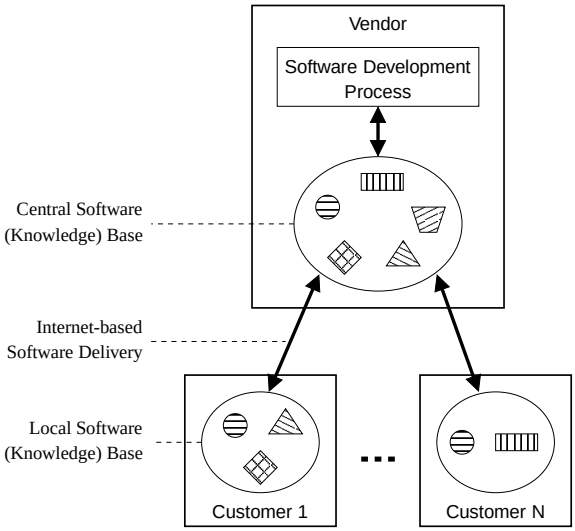

Figure 1: Software (knowledge) delivery model proposed by the Deliver project.

An important aspect of the Deliver model is the network-based delivery of both software and knowledge about the software. Network-based software (knowledge) delivery refers to use of computer networks, such as the Internet, to deliver software, as either full packages or incremental updates, and knowledge about the software, such as versioning and dependency information to customers. Network-based software (knowledge) delivery lowers the overhead of providing new releases, shortens the release phase of a product, and provides customers and vendors with insight in the delivery process.

Figure 1 shows our network-based software delivery model with the ISKB. In this model, a software vendor develops software, according to its own software development process, resulting in artifacts and their metadata, which are subsequently stored in the CSKB. The figure shows a CSKB with five deliverable artifacts that can be downloaded by customers. The figure also shows N customers that have downloaded these deliverables into their LSKBs, including their metadata. Customer 1 has a configuration with two artifacts and Customer N has a configuration with three.

To build the ISKB, we face challenges in four areas. The first area is the extraction of information. The question here is what basic facts to gather about the software and its processes and how to gather these facts. The second area deals with the inference of knowledge. The question here is how

[1] SWEBOK: The Software Engineering Body of Knowledge

to extract useful "higher-level" information from the basic facts. The third area deals with the representation of the basic facts and higher-level information. The question here is how to store both the basic facts and higher-level information in an efficient and effective way. The fourth area revolves around the application of the higher-level information. The question here is how to apply this information to solve real-life business needs. In our view software knowledge comprises both the basic facts and higher-level information.

3. Case Study Design

The overall goal of the case study of ChipSoft was to provide insight and help us deal with the four problem areas of our ISKB. To structure the case study, we formulated the following more-specific research goals.

G1 To describe the current development, release, and deployment activities of ChipSoft.

G2 To relate these activities to the initial Deliver model.

The description would cover the procedures, techniques, and tools used by ChipSoft and its customers and the context in which they were used. This description would include the problems and limitations of these procedures, techniques, and tools. The analysis would answer the question what improvements and extensions would be needed in the Deliver model. Of particular interest to both the Deliver project and ChipSoft was determining to what extent the model could help ChipSoft solve existing release management problems or open up new business opportunities.

ChipSoft is relevant to our research project since its product CS-ECIS, a health-care information system, deals with an application domain with specific characteristics. Examining ChipSoft therefore allows us to see the release management activities of a company that is significantly different from previous case studies in other application domains [12, 13]. For instance, since the majority of customers of ChipSoft are hospitals, ChipSoft has relatively few customers but many workspaces per customer. This is a different kind of customer base than the customer base of Exact Software [12] and Planon [13].

From the two research goals, we derived the following four more specific research questions:

Q1 What are the development, release, and deployment activities for CS-ECIS?

Q2 What tools and techniques are used by ChipSoft to support these activities?

Q3 How do these activities, tools, and techniques relate to the Deliver model?

Q4 What aspects of the Deliver model should be improved or extended?

To answer the research questions as precise and complete as possible, we created a case study protocol to guide our examinations and a case study database to store intermediate results. Furthermore, to achieve correctness of the end-result, we cross-checked our findings where possible. As a final check the work was critically reviewed by ChipSoft.

During the case study, we collected information to answer the four research questions and achieve the two research goals. Since the case study was mostly descriptive and exploratory [24], this information was largely of a qualitative nature. This information was gathered from the following sources:

- Interviews
- Software examination
- Document study
- Direct observations

During the case study, we spoke to the following types of personnel at ChipSoft.

- Development personnel
- Release personnel
- In-house consultants
- Management

Even though the system administrators and end users of CS-ECIS could provide useful information, they were not approached due to practical and business-related considerations. We do not expect this exclusion to have a strong influence on the case study since most customer-related information was found indirectly from the in-house consultants of ChipSoft.

4. Case Study Results

4.1. ChipSoft

ChipSoft is a Dutch medium-size company that provides comprehensive ICT solutions in the health-care domain. Its main business activities are the production and sale of a health-care information system, the customization of this product for specific customers, and the reselling of all required third-party hardware and software. ChipSoft currently has a customer base of approximately 40 hospitals spread over six countries, with most customers located in the Netherlands.

ChipSoft was founded in 1986 and has been growing ever since. ChipSoft currently employs approximately 100 employees. ChipSoft started with producing applications for the automation of the primary care process(es) of medical specialists. Over the following years, ChipSoft extended and

Table 1: Approximate number of employees per department.

Research & Development	33
Implementation & Support	42
Marketing & Sales	17
Administrative Support	8
Total	100

integrated its product range with support for other processes in the medical field, resulting in the development of a comprehensive hospital information system, called ZIS, in 1994. In the autumn of 2001, it released the first release of its current main product: the Electronic Care Information System (CS-ECIS).[2]

ChipSoft currently consists of four departments: Research & Development (R&D), Implementation & Support (I&S), Marketing & Sales (M&S), and Administrative Support. Table 1 shows how the ChipSoft's 100 employees are distributed over the four departments. The technological side of the activities are spread over R&D and I&S. The R&D department consists of thirty-three software engineers, and is responsible for the development and maintenance of the core application functionality. The I&S department, in contrast, consists of forty-two consultants, and is responsible for the implementation of the application at specific customers, including customer-specific customizations. I&S is also responsible for providing tutorials on the use of ChipSoft products. Both departments are subdivided to deal with different health-care problem domains.

4.2. The Electronic Care Information System

ChipSoft's current product is its "Electronic Care Information System" (CS-ECIS) that supports the various workflows in a hospital. It can be used to support both the direct, care activities, such as diagnosis and treatment, and the more indirect, management activities, such as planning and financial settlement. For these functions, CS-ECIS stores and processes information about both patients and the various resources (e.g., beds or ORs) in a hospital. CS-ECIS is used by a diverse group of end users, such as, secretaries, nurses, doctors, and managers.

To enlarge its versatility, CS-ECIS consists (conceptually) of a small shell application that uses a set of extension modules that implement the main functionality. ChipSoft currently supports twenty-two modules. The different modules support the different workflows found in a hospital. Since customers can choose which modules to buy, different customers will use different module configurations. CS-ECIS

[2]The official product name in Dutch is: Electronisch Zorg Informatie Systeem (CS-EZIS).

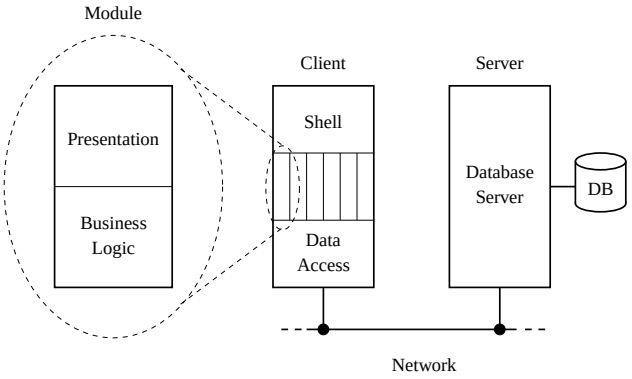

Figure 2: The CS-ECIS architecture consist of clients and a server, where the client is divided into a shell application with a variable number of extension/implementation modules.

is therefore more accurately described as a software product family than as a single software product.

As shown in Figure 2, CS-ECIS uses a standard client-server architecture. ChipSoft develops, however, only the client-side software itself, and simply resells a standard database server (e.g., MS SQL server). Within this architecture, the application shell and its extension modules are located only at the client. Every CS-ECIS module has a three-layer internal architecture: on top a presentation layer with GUI code, an intermediate layer with the business logic, and the bottom layer with database access code. The top and intermediate layer provide the functionality for specific workflows, while the bottom layer provides a generic database abstraction shared by all modules. CS-ECIS currently stores over 1,000 tables in its database and is built from approximately 2.2 million lines of code.

All ChipSoft software is targeted toward Microsoft operating systems, for instance, Windows 98 for the client application and Windows 2000 for the server. This server can also run on the Novell operating system. Apart from the operating system, ChipSoft places few other restrictions on the platform used. Hospitals are free to use whatever system administration support tools they desire. However, hospitals frequently use a terminal server set-up (e.g., a Citrix server), where all software is installed and executed on a single, heavy-weight server and light-weight client machines (i.e., terminals) can log into that server. Such a setup is popular since it reduces the software maintenance overhead.

4.3. Support for Variability and Extensibility

To allow the application of CS-ECIS in a wide range of hospitals, CS-ECIS is highly configurable. We can distinguish the following types of variability in CS-ECIS:

- Activation of modules

- Per-module configuration options
- User profiles
- GUI modifications and additions
  - Layout and contents of interactive screens
  - Layout and contents of static reports
- Functionality modifications and additions
  - User-defined fields
  - User-defined expressions
  - User-defined actions
- Customer-specific modules

Since ChipSoft always distributes the *complete* set of modules of CS-ECIS to a customer, the modules actually bought by a customer need to be activated before use. ChipSoft employs a simple licensing scheme with respect to activation, where some parts of the software are licensed to (i.e., activated for) the whole organization (e.g., hospital) and other parts are licensed per workspace. The duration of a license commonly ranges from five to eight years. Licensing is partially enforced through module activation records in the database, which are checked at various places in the software. Licensing is, however, mostly enforced through the organization's dependence on the support (i.e., consultancy) contract with ChipSoft.

To enhance applicability, modules have their own configuration options, for instance, to support different third-party applications, use different types of patient numbers, or perform some function differently as a matter of policy. For example, different hospitals use different types of patient numbers (e.g., 7, 9, or 11 digits), and a hospital can select the type it uses. To enable different end-users to see only those parts of CS-ECIS they use, the application manager can create user profiles. A user profile describes the modules an end-user is allowed to use, and different users can be assigned different profiles.

A large source of variability is the modification of and additions to the graphical user interface (GUI) of CS-ECIS and its basic functionality. To support the GUI customization, CS-ECIS contains a GUI builder. This GUI builder allows end-users to add and modify GUI controls (i.e. widgets), such as, labels, database records and fields, computed values, buttons, and menu's. Modifications to the GUI are stored in the database as XML-encoded data.

When the information that CS-ECIS stores, computes, or shows by default, is incomplete or not usable by a customer, CS-ECIS can be configured (i.e., extended) to provide the desired situation. For instance, different fields of a table can be extracted or a different value can be computed. If that is not enough, the end-user can also add new fields to the tables in the database. These new fields can subsequently be shown and modified in the GUI. If an end user is dissatisfied with the default interaction with CS-ECIS, she can add and modify the behavior of the buttons and menu items in the GUI. For example, a user can specify actions to be performed on the database, and bind these actions to a button in the user interface.

When a customer needs functionality that cannot be created using standard configuration options in CS-ECIS, a customer-specific module is developed by the R&D department, and this module becomes part of the main code base. The module is then distributed to all customers during a regular release, but is activated only for that particular customer. The customer-specific module can obviously also be activated for other customers if it proves to be more generally applicable. Note that CS-ECIS does not use or support variant modules, that is, interchangeable modules that provide similar functionality.

4.4. Release and Deployment

Twice a year (i.e., in spring and in autumn), ChipSoft creates a new *release* of the CS-ECIS software. Between releases, ChipSoft sometimes provides one or more service packs. A *service pack* is created whenever the software has significantly changed, these changes are required immediately by customers, and the next release is too far off. A service pack is actually functionally the same as an ordinary release, but is named differently mostly for marketing reasons. Since a service pack is a full update (i.e., not incremental), it includes all previous service packs and the release on which it is based. To receive support, customers are (contractually) allowed to be at most two releases behind the current release. ChipSoft supports a separate parallel release, called a *feature pack*, that contains experimental features implemented for specific organizations (see also Section 4.5.1).

A release (or service pack) consists of executables, DLLs, resource files, and release notes. When the release requires changes to the database schema's in the server, a special database conversion program is also included. As mentioned before, even though CS-ECIS is implemented and marketed as a collection of 22 independent modules, it is always delivered as a whole, with all modules included. The actual distribution of releases and service packs is done using CD-ROMs and the postal service.

When a customer reports a critical defect, ChipSoft repairs this defect using a special kind of software distribution, called a hotfix. A *hotfix* is an incremental update that consists of only those files that need to be replaced in the installed release or service pack to solve the problem. While the replaced files can be of any types, hotfixes usually involve only DLL files. To give customers quick access to hotfixes, hotfixes are distributed using a password-protected website. Some problems cannot be fixed using a hotfix and require a full update using a service pack.

A hotfix is intended to deal with only a single problem, and therefore does not include previous hotfixes that dealt with other modules. Customers are expected to install only those hotfixes that are strictly needed, but otherwise wait for the next regular release or service pack. Customers thus frequently have only some (or none) of the hotfixes installed. Since most dependencies between DLLs are within a single module, a hotfix includes all the DLLs in the module that have changed since the previous release or service pack. A hotfix thus includes previous hotfixes that dealt with the *same module*, thereby avoiding DLL dependency problems. Currently, software developers at ChipSoft have to manually track which deliverable files (i.e., DLLs) have changed and thus need to be replaced by the hotfix.

ChipSoft refers to its software distributions in two ways: a structured, customer-friendly distribution name and an absolute build number. The format of the distribution name depends on distribution type. A release is named using a pair of numbers (i.e., "4.x"). Thus far, all releases of CS-ECIS have started with major version "4", and only the minor number of the releases have changed. The release for autumn 2004 is named "4.6". A service pack is named using the release name on which it is based and a number (i.e., "4.x SP y"). Service packs are numbered consecutively and this number is reset on every new release. A hotfix is named using the release or service pack name on which it is based and a number (i.e., "4.x HF z" or "4.x SP y HF z"). Hotfixes are also consecutively numbered, and the hotfix number is reset on every new release or service pack.

The format of the absolute build number is always a 4-tuple (i.e., "4.r.s.b") consisting of the major number of the release (thus far always "4" for CS-ECIS), the minor number of the release (r), the service pack number (s), and the relative build number (b). To indicate a build number for a normal release, the service pack number is set to "0" (e.g., "4.5.0"). The relative build number is reset only with a release and not with a service pack or hotfix. Using the label/tag concept of its version management system, ChipSoft can easily map a distribution name to the absolute build number of the originating build run. This absolute build number refers to the build run that created the software distribution.

To identify the software installed at a particular customer, ChipSoft stores both its base version, that is, the name of the installed release or service pack, and a list of zero or more hotfixes, that is, the hotfixes installed by this particular customer, in the database of the customer. The following is an example of this kind of installed software identification:

```
4.5 SP1         (4.5.1.78)
4.5 SP1 HF1     (4.5.1.85)
4.5 SP1 HF3     (4.5.1.98)
```

This customer has installed service pack "1" for release "4.5" together with hotfixes "1" and "3". The service pack was created in build "4.5.1.78" and the hotfixes were created in build "4.5.1.85" and "4.5.1.98".

ChipSoft offers no support for the internal deployment (i.e., distribution and installation) of CS-ECIS software in a hospital. Instead, hospitals are themselves responsible for the internal deployment of CS-ECIS, and are expected to use, for instance, COTS products. The main reason for not providing this type of support, is that there are many ways in which a hospital can perform the internal deployment, such as system image distribution, manual installation, or installation on a central server, but these methods all strongly depend on the structure and properties of the local computing environment. A capable ICT department with capable application managers, is therefore required to support CS-ECIS.

4.5. Development and Maintenance

ChipSoft develops CS-ECIS using the Delphi programming language, and the main development tasks of the software developers, such as editing, compiling, and debugging, are supported by the Delphi integrated development environment (IDE). To implement CS-ECIS, ChipSoft uses several third-party software products. The most obvious external component is the database server, which can be either the Microsoft SQL Server or the Advantage Database Server. To deal with medical images, for instance, from X-ray machines, ChipSoft uses PACS (picture archiving and communication system) software. This software is provided by the hardware vendor. Finally, to create reports from the database for management purposes, the report builder software from Digital-Metaphors is used.

4.5.1. Version Management

For version management, ChipSoft uses Microsoft's Visual SourceSafe (VSS). In its source code repositories, ChipSoft uses mostly a file locking policy (i.e., pessimistic concurrency control). Locking source files is not perceived as a bottleneck since the development work is distributed over the developers in disjoint subsets. To combine concurrent *development* of a new release with *maintenance* on old releases, ChipSoft uses a simple branching scheme, shown in Figure 3. The upcoming release (Release 4.7) is stored in the trunk of the repository and the most recent releases (Releases 4.4–4.6) are available in branches.

Regularly, ChipSoft desires to develop special features that require a longer running effort but that unfortunately might also interfere with normal development on the trunk. To insulate the trunk from these changes, a separate *feature pack branch*. This branch is created on the latest release branch and will be merged back to the trunk after the next release. To keep the version tree manageable, only a single feature pack can be active at a time. As a consequence, when multiple special features are developed concurrently, the fea-

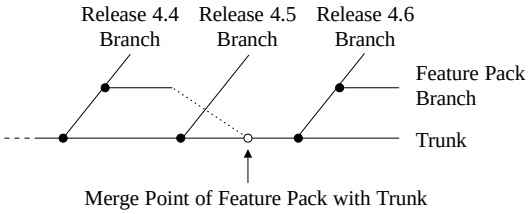

Figure 3: Version Tree.

ture pack will contain the results from each special-feature development effort. As an example, Figure 3 shows two feature packs, one on Release 4.4 and one on Release 4.6. The changes from the Release 4.4 feature pack, have been integrated into Release 4.6, while the efforts of the feature pack of Release 4.6 are still ongoing.

To simplify version management, there is no independent version management for the various modules. Instead, ChipSoft considers only revisions of the CS-ECIS application as a whole, including database schema's. This means that only the client and server within a single release are guaranteed to be compatible, and a customer cannot independently update the client. As a consequence, consistency between artifacts can be maintained using their version management system.

ChipSoft does not use a (formal) product description that describes the internal and external dependencies of the modules (or DLLs) that make up CS-ECIS. As a consequence, these internal dependencies, for instance caused by inter-module method calls, are silently ignored. External dependencies are dealt with only informally. Other companies simply keep ChipSoft informed of major changes to the interfaces of their hard- and software. Care must also be given to the dependency of the user-defined expressions (see Section 4.3) on the expression language implemented by a particular release of the client. This dependency problem is dealt with by keeping the expression language backward compatible.

4.5.2. Build and Release Process

The global development and release flow is shown in Figure 4. Developers work at their workstations and commit new and modified artifacts to the central source code repository. To create global builds for testing and release, ChipSoft uses a dedicated build server from Atozed. The build compiles the sources stored in the repository and creates a directory structure with the release artifacts. The directory can subsequently be used by the Install Shield tool to create an installation image for the installation CD-ROM.

This build server provides a clean build environment that ensures that the program does not dependent on a specific workspace configuration of a developer. A global build is specifically requested whenever the interface of a DLL has changed and an integration test is needed, and is started manually, usually twice per day. To avoid having to rebuild everything themselves, software developers can download the latest DLLs from the build server for use in their private IDEs.

4.5.3. Testing

ChipSoft uses no formal techniques to verify the correctness of their software. Instead, correctness is maintained only through testing. The build server performs a simple integration test by providing a clean build environment, which ensures the reproducibility of the build. Furthermore, a formally described test phase is entered for every release (i.e., twice a year). This testing is done by hand by the I&S department, and the test use cases are described in a manual. Since these tests require a significant amount of effort (about two weeks), ChipSoft performs these tests only on releases, and relies on its software engineers to thoroughly test their modifications for service packs and hotfixes. ChipSoft does not use any automatic unit, smoke, or integration test.

Since the various hospitals that use CS-ECIS can have very different hard- and software configurations, ChipSoft cannot test its new releases in each and every configuration. However, since the CS-ECIS is mission critical, hospitals cannot simply install the software and hope for the best. Instead, to deal with this potential incompatibility problem, each hospital needs a local test setup where it can safely test the new software in its particular environment. Furthermore, this test setup allows the application managers to experiment with new features without interfering with normal hospital operations.

4.5.4. Internal Communication

To administrate defect information, ChipSoft uses the open source defect tracker Mantis. This system is for internal use only. There is currently no strong integration between ChipSoft's version management system and its defect tracking system (Mantis). Log messages in the VSS do refer to defect numbers in Mantis, but defect tracking entries do not mention the version in which the defect was fixed. To record defect reports and enhancement requests from customers, a separate, locally developed, customer relationship management (CRM) system is used. This system also records all customer information, including contract information.

To avoid problems with major changes to the software, such as adding a new module, ChipSoft uses documented procedures for these types of changes, described on the internal website. For the most part, however, the source code and more-experienced programmers are considered the main sources of information. To improve the overall knowledge of its programmers of the software, ChipSoft organizes an internal seminar twice a year.

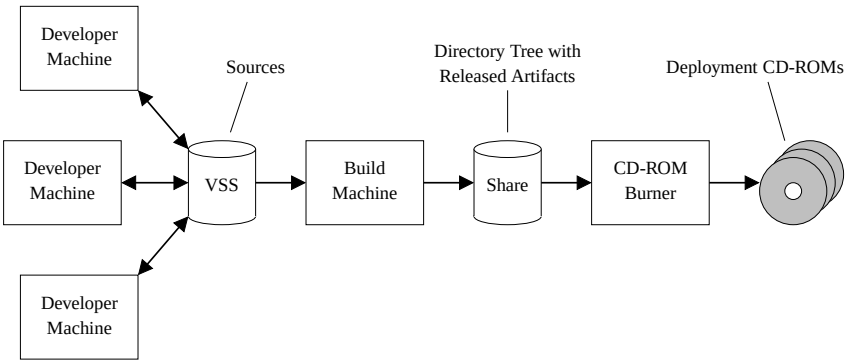

Figure 4: Development and release workfbw in ChipSoft.

5. Discussion

We started our case study of CS-ECIS and ChipSoft with the overall goal of gaining insight in the release management activities of a real-life company. When considering what we found, an obvious observation is the existence of major differences between the Deliver model and ChipSoft's modus operandi. The four most pronounced differences are the use of modularization, the use of configuration and variability, the use of network-based software deployment, and the explicit management of software knowledge.

5.1. Modularized Software

The first difference between ChipSoft's activities and the Deliver model concerns the use of modularization. To gain flexibility, the Deliver model expects the use of modularization in all phases of the software life cycle, such as design, implementation, testing, release, delivery, and installation. At ChipSoft, in contrast, modularization is used solely during design and implementation and as a marketing technique. For instance, all modules of CS-ECIS are versioned as a whole and all modules are always present in every installation.

5.2. Configurable Software

The second difference concerns the use and support of variability and configurability. To gain further flexibility, the Deliver model expects the use of configurable modules, variable configurations of modules, and variable architectures [22]. This support enables the creation of product families, product lines, and product populations through module composition. In the model, variability can be bound at the various times, for example, release, installation, or runtime. At ChipSoft, variability and configurability is achieved only at the product feature level through its use of a configurable and extensible GUI, database, and bindings. Since ChipSoft does not use variability through (dynamic) application composition, variability is bound at installation and runtime.

5.3. Network-based Software Deployment

The third difference between ChipSoft's activities and the Deliver model concerns the use of computer networks for the deployment of software, using protocols that take versioning information into account. The Deliver model expects the use of computer networks, such as the Internet, to deliver software as either full packages or incremental updates, to decrease the overhead of providing new versions and shortens the release phase of a product, preferring bits over atoms. ChipSoft currently uses network-based software delivery only for its hotfixes where short release cycles are important. Furthermore, the protocols used are generic WWW protocols without support for versioning.

5.4. Software Knowledge Management

The final (major) difference concerns the active and explicit use and management of software knowledge. To automate the release, and relate development and deployment, activities, and gain insight in them, the Deliver model promotes the active gathering and processing of software knowledge. This knowledge includes knowledge about what is installed at various customers. These activities require a (semi-)formalized model of both the software product and its processes. In contrast, ChipSoft currently manages only a limited amount of information (e.g., build information and defect tracking), and this information is not dealt with in an integrated fashion. Most information about the product and processes are dealt with only informally. Furthermore, ChipSoft only has limited knowledge of the software installed at their customers' sites.

5.5. Lessons Learned

ChipSoft is a successful company that has grown steadily over the years. Given this success and the clear differences

between the Deliver model and ChipSoft, these differences clearly do not represent strong requirements in its application domain. One reason why the flexibility of the Deliver model might not be required, suggested by ChipSoft, is that the health-care application domain is very conservative when it comes to managing ICT. As software changes might interfere with day-to-day operations and government regulation change regularly requiring software changes anyway, other changes are usually undesirable. Regular small change to the software are all that is needed and desired.

The main lesson learned for the Deliver model during the case study is the fact that the model is not universally usable. There are software application domains that do not need the support it provides. The question then remains what applications domain do require the support of the Deliver model and its Intelligent Software Knowledge Base. An application area where much modularity is required and software evolves rapidly is embedded software. Further investigation will therefore include this area.

6. Related Work

Before the introduction of component-based software engineering (CBSE) and network-based software delivery, release management was barely recognized as as a separate field within software configuration management, as evident from [5, 20]. However, their introduction has made comprehensive release management both challenging and essential.

Several research systems have taken the challenge to support component-based software products created by communities of distributed developers. Examples of such systems are SRM [21], Software Dock [9], and the online package base [6]. Others have considered proper ways of modeling the deployment process. Of particular interest in this research is the problem of specifying component dependencies and configuration, as dealt with in the Deployable Software Description (DSD) [11], and Open Software Description (OSD) and Management Information Format (MIF) [10]. An extension of this work also considers their relation to feature models [22]. An other issue to consider is the proper modeling of the component life cycle, such as in [3, 14].

It is well known that the development and management of software is knowledge intensive [19], and that managing this knowledge can be profitable but difficult [16]. To deal with this difficulty, various knowledge bases have been proposed that can roughly be divided into two categories. The first category deals with storing knowledge about the problem-solution space. An example of such as system is ConceptBase [15]. The second category deals with storing knowledge about the software assets themselves. Some systems in this category focus on software development focusing on reuse [4] or consistency requirements [17]. The similarity of these systems with product data management (PDM) systems has lead to research efforts to come to some form of integration [8, 18].

7. Conclusion

This paper describes the results of a case study we performed of the release management of CS-ECIS, a health-care information system developed by the Dutch software vendor ChipSoft. This case study had two goals. The first goal was to describe the release management and related development and deployment activities of ChipSoft. The second goal was to compare these activities with the initial Deliver model, including the Intelligent Software Knowledge Base (ISKB).

The main conclusion from this case study is that there is currently no business case for introducing an ISKB at ChipSoft since ChipSoft's requirements are too different from the Deliver model. Subsequently, little insight was gained with respect to the challenges in designing an ISBK. Therefore, different companies have to be found where we can gain this insight. The case study does provide the possibility of comparing this case with the Exact Software [12] and Planon [13] cases, as part of future work.

Acknowledgments

We would like to thank Ernst ten Damme, Chris Endhoven, Bertus Buitenhuis, and Robert Hardholt of ChipSoft for the contributions made to this case study.

References

[1] E. Bailey. *Maximum RPM*. 1997.

[2] G. Ballintijn. A case study report on the development, release, and deployment processes of chipsoft. Technical Report SEN-E0506, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, Apr. 2005.

[3] A. Carzaniga, A. Fuggetta, R. S. Hall, A. van der Hoek, D. Heimbigner, and A. L. Wolf. A characterization framework for software deployment technologies. Technical Report CU-CS-857-98, Department of Computer Science, University of Colorado, Boulder, CO, USA, 1998.

[4] P. Constantopoulos, M. Jarke, J. Mylopoulos, and Y. Vassiliou. The software information base: A server for reuse. *VLDB Journal*, 4(1):1–43, 1995.

[5] S. Dart. Concepts in configuration management systems. In *Proceedings of the 3rd International Workshop on Software Configuration Management*, pages 1–18. ACM Press, June 1991.

[6] M. de Jonge. Source tree composition. In *Proceedings: Seventh International Conference on Software Reuse*, volume 2319 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[7] J. Dunagan, R. Roussev, B. Daniels, A. Johnson, C. Verbowski, and Y.-M. Wang. Towards a self-managing software patching process using black-box persistent-state manifests. In *Proceedings of the International Conference on Autonomic Computing (ICAC'04)*, pages 106–113, New York City, NY, USA, May 2004.

[8] J. Estublier, J.-M. Favre, and P. Morat. Toward SCM / PDM integration? In *Proceedings of the 8th Workshop on Software Configuration Management (SCM 8), LNCS 1439*, pages 75–94, July 1998.

[9] R. S. Hall, D. Heimbigner, A. van der Hoek, and A. L. Wolf. An architecture for post-development configuration management in a wide-area network. In *Proceedings of the 17th International Conference on Distributed Computer Systems (ICDCS)*, pages 269–278, Baltimore, MD, USA, May 1997.

[10] R. S. Hall, D. Heimbigner, and A. L. Wolf. Evaluating software deployment languages and schema. In *Proceedings of the 1998 International Conference on Software Maintenance (ICSM)*, pages 177–185, Bethesda, MD, USA, Nov. 1998.

[11] D. Heimbigner, R. S. Hall, and A. L. Wolf. A framework for analyzing configurations of deployable software systems. In *Proceedings of the Fifth IEEE Int'l Conference on Engineering of Complex Computer Systems*, pages 32–42, Las Vegas, NV, USA, Oct. 1999.

[12] R. Jansen, G. Ballintijn, and S. Brinkkemper. Software release and deployment at Exact: A case study report. Technical Report SEN-E0414, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, Sept. 2004.

[13] S. Jansen, G. Ballintijn, and S. Brinkkemper. Release and deployment at Planon: A case study. Technical Report SEN-E0504, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, The Netherlands, Mar. 2005.

[14] S. Jansen, S. Brinkkemper, and G. Ballintijn. A process model and typology for software product updaters. In *Proceedings of the 9th European Conference on Software Maintenance and Reengineering (CSMR 2005)*, Manchester, United Kingdom, Mar. 2005.

[15] M. A. Jeusfeld, M. Jarke, H. W. Nissen, and M. Staudt. Conceptbase: Managing conceptual models about information systems. In P. Bernus, K. Mertins, and G. Schmidt, editors, *Handbook of Information Systems*, pages 265–285. Springer-Verlag, 1998.

[16] P. Klint and C. Verhoef. Enabling the creation of knowledge about software assets. *Data Knowl. Eng.*, 41(2-3):141–158, June 2002.

[17] B. Meyer. The software knowledge base. In *Proceedings of the 8th International Conference on Software Engineering*, pages 158–165, London, United Kingdom, Aug. 1985.

[18] A. Persson-Dahlqvist, I. Crnkovic, and M. Larsson. Managing complex systems - challenges for pdm and scm. In *Proceedings of the Tenth International Workshop on Software Configuration Management (SCM-10,ICSE-23)*, Toronto, Canada, May 2001.

[19] P. N. Robillard. The role of knowledge in software development. *Communications of the ACM*, 42(1):87–92, Jan. 1999.

[20] J. A. Scott and D. Nisse. Software configuration management. In *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, chapter 7, pages 103–119. IEEE, May 2001.

[21] A. van der Hoek and A. L. Wolf. Software release management for component-based software. *Software — Practice and Experience*, 33:77–98, 2003.

[22] T. van der Storm. Variability and component composition. In *Proceedings of the Eighth International Conference on Software Reuse (ICSR-8)*, Madrid, Spain, July 2004.

[23] R. van Ommering. Building product populations with software components. In *Proceedings of the 24th international conference on Software engineering*, pages 255–265, Orlando, FL, USA, May 2002.

[24] R. K. Yin. *Case Study Research: Design and Methods*, volume 5 of *Applied Social Research Methods Series*. Sage Publications, Inc., Thousand Oaks, CA, USA, 3 edition, 2003.