Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

SEN

Software Engineering

Software ENgineering

A case study report on the development, release, and deployment processes of ChipSoft

G. Ballintijn

REPORT SEN-E0506 APRIL 2005

CWI is the National Research Institute for Mathematics and Computer Science. It is sponsored by the Netherlands Organization for Scientific Research (NWO).
CWI is a founding member of ERCIM, the European Research Consortium for Informatics and Mathematics.

CWI's research has a theme-oriented structure and is grouped into four clusters. Listed below are the names of the clusters and in parentheses their acronyms.

Probability, Networks and Algorithms (PNA)

Software Engineering (SEN)

Modelling, Analysis and Simulation (MAS)

Information Systems (INS)

P.O. Box 94079, 1090 GB Amsterdam (NL)
Kruislaan 413, 1098 SJ Amsterdam (NL)
Telephone +31 20 592 9333
Telefax +31 20 592 4199

ISSN 1386-369X

A case study report on the development, release, and deployment processes of ChipSoft

ABSTRACT
This report contains the results of a case study we performed of the Dutch software vendor ChipSoft. This case study is part of the Deliver research project, and it focused on ChipSoft's release, delivery, and deployment activities. We performed the case study to gain insight into these activities which, in turn, would enable us to propose new ways to ease their effort and reduce their risks. The case study consisted of recording the development, release, delivery, and deployment activities of ChipSoft and comparing these with the activities of the model proposed by the Deliver project. The description and comparison enabled us to both evaluate the Deliver model and propose improvements to the activities of ChipSoft.

2000 Mathematics Subject Classification: 68N99
1998 ACM Computing Classification System: D2.9, D2.7
Keywords and Phrases: ChipSoft; release management; version management; product data management; software configuration management; software deployment; software development
Note: This work was carried out under project SEN1 - Deliver.

A Case Study Report on the Development, Release, and Deployment Processes of ChipSoft

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

Abstract

This report contains the results of a case study we performed of the Dutch software vendor ChipSoft. This case study is part of the Deliver research project, and it focused on ChipSoft's release, delivery, and deployment activities. We performed the case study to gain insight into these activities which, in turn, would enable us to propose new ways to ease their effort and reduce their risks. The case study consisted of recording the development, release, delivery, and deployment activities of ChipSoft and comparing these with the activities of the model proposed by the Deliver project. The description and comparison enabled us to both evaluate the Deliver model and propose improvements to the activities of ChipSoft.

1 Introduction

The release, delivery, and deployment 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 or variant of the application. Furthermore, these enterprise applications frequently consist of many (software) components that depend on each other to function correctly. On top of that, these components will evolve over time to answer the changing needs of customers. As a consequence, the release, delivery, and deployment of enterprise applications takes a significant amount of effort and is frequently error-prone.

The goal of the Deliver project is to ease the software release and deployment effort and reduce its risks [10]. We propose to achieve both by managing explicitly all the knowledge about the software. To start with, by managing this software knowledge explicitly, a software vendor can improve the update process of its software. For instance, the creation of incremental updates can be simplified and consistency requirements can be enforced. Furthermore, the explicit management of software knowledge also enables the evaluation of "what if" scenarios, such as, what will happen to the software configuration of a customer, when she updates a certain application component? These evaluations improve the risk assessment of the deployment process, which, in turn, can improve the quality of customer support.

Managing software knowledge is only part of a software delivery solution. The software still needs to be delivered to customers. To ease the delivery effort and reduce its duration, we desire support for the delivery of software via the Internet, both as full packages and incremental updates [9]. Fortunately, the explicitly managed software knowledge can also be used to enhance the delivery of software. Specifically, it can be used to compute the difference between the existing software configuration at a customer and the desired configuration. This difference can then be used to create the required updates. To deliver the software update to the customer, we will have to develop specialized delivery methods and protocols.

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

issues occur. To gain this insight, we use the case study research method. Additionally, the case studies offer the companies involved an external assessment of their release, delivery, and deployment activities. Based on these assessments, we can propose improvements and extensions to the current practices of the companies.

After performing case studies at the Dutch software vendors Exact Software [5, 7] and Planon [8, 12], we decided to examine the Dutch software vendor ChipSoft [2]. 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 release, delivery, and deployment activities of ChipSoft and the tools and techniques used to support them. The second goal was to compare these activities, tools, and techniques to the theoretical models we developed in the Deliver project. The findings of the case study are described in this report.

The rest of this report is structured as follows. In Section 2, we describe the approach we used during the case studies. Section 3 and 4 give descriptions of ChipSoft and the products it sells, respectively. In Section 5, we describe the software release, delivery, and deployment processes and tools used to support them. Similarly, Section 6 describes the software development and maintenance processes and tools used to support them. In Section 7, we evaluate our findings with respect to the Deliver model, and in Section 8 we draw our conclusions.

2 Research Approach

2.1 Research Objectives and Questions

The case study of ChipSoft had two research goals.

G1 *To describe the current release, delivery, and deployment activities of ChipSoft.*
This description would cover the procedures, techniques, and tools used, including the context in which they were used, both at ChipSoft and its customers. An important aspect of this goal was the identification and description of the problems and limitations of these procedures, techniques, and tools.

G2 *To relate these activities to the activities of the Deliver model.*
This analysis would answer the question of whether the Deliver model could be applied to ChipSoft and, if so, what improvements and extensions would be possible. Of particular interest to both the Deliver project and ChipSoft was determining to what extent the model could help ChipSoft solve existing release, delivery, and deployment problems or open up new business opportunities.

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

Q1 What are the release, delivery, and deployment processes for ChipSoft products?

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

Q3 How do these processes and techniques relate to the processes and techniques proposed by the Deliver model?

Q4 What processes and techniques can be improved or extended, both at ChipSoft and in the Deliver model?

2.2 The Deliver Model

The Deliver model for software release, delivery, and deployment can be divided into two parts [10]:

- The intelligent software knowledge base (ISKB)
- Network-based software delivery

The intelligent software knowledge base plays a central role in the software life cycle in that it supports development, release, and deployment activities, both at the vendor site and at the customer site. Conceptually, the ISKB can be considered a cross between a software configuration management (SCM) system and a (software) product data management (PDM) system [4]. Since the focus of our research is on release, delivery, and deployment processes, we consider the development process only in so far it influences these other processes. In contrast, earlier work by Meyer [11] describes an ISKB that is focused primarily on the development of software.

The ISKB 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, delivery, and deployment processes, each pose different challenges to the ISKB, requiring different kinds of information. 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, 13] and the on-line dependency analyzer (OA) in [3].

The following list summarizes the main features of an ISKB.

- *Software Update and Evaluating "What if" Questions*
  The ISKB will implement mechanisms to generate a new configuration for a customer depending on the deliverables the customer already has installed. This functionality will require the ability to query both the LSKB at the customer site and the CSKB at the vendor site and combine their information to check for conflicting revisions or variants of deliverables. If no conflicts exist, a list of deliverables that need to be installed can be generated, enabling an efficient incremental update process.

- *Customer-installation Maintenance*
  When the software is delivered to the customer, a tool is needed that deploys, configures, and possibly builds the software. To recover from faulty software updates, this tool should be able to rollback a software configuration to a previous state. This feature depends, unfortunately, heavily upon the underlying platform, making its implementation potentially difficult. Another important aspect of such a deployment tool is support for Web-based software delivery.

- *Integration with Version Management System*
  The ISKB needs a versioned repository to store all relevant artifacts. This repository should be able to concurrently store different revisions and variants of a single artifact, for instance, a source code file. We are, however, not planning to implement yet another version management system, but instead expect to integrate the ISKB with an existing version management system.

- *Integration with Build Systems*
  The knowledge stored in the ISKB can also be used to guide the automatic building of executables. To support this capability, the dependency relationships between source files and deliverables need to be added to the ISKB when a product is designed. This build dependency information enables the ISKB to maintain consistency between original and derived artifacts.

Figure 1 shows the original network-based software delivery model, as proposed in the Deliver project proposal [10]. 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 (including metadata) into their LSKBs. Customer 1 has a configuration with two artifacts and Customer N has a configuration with three.

Preliminary discussions with ChipSoft have shown, however, that the Deliver model might not be directly applicable to ChipSoft since the model assumes a customer base consisting of many customers with only a single workspace while the ChipSoft customer base consists of only a few customers but with many different workspaces. We therefore extended the original delivery model to include a more complex

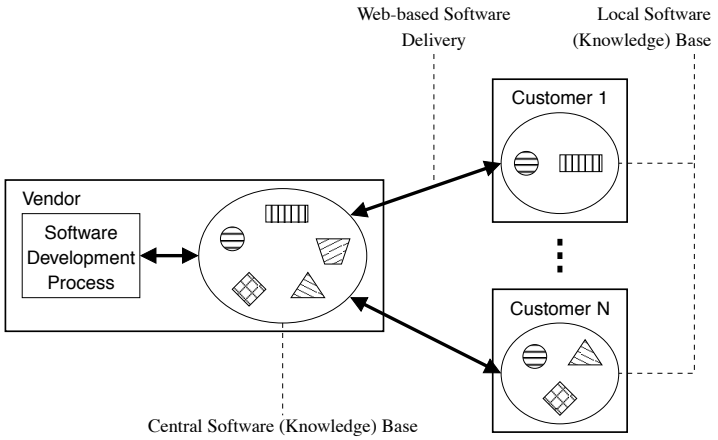

Figure 1: Original software deliver model proposed by the Deliver project.

customer site. Figure 2 shows this extended model, focusing on a single customer C with N workspaces (e.g., PCs). The figure also shows that the vendor site has remained the same. The customer site has a single *Site-wide Software Knowledge Base* (SSKB) and a LSKB at every workspace. The SSKB can be used, for instance, to cache artifacts and to (partially) configure them. Figure 2 shows an SSKB on a host called Repository containing four artifacts, Workspace 1 with a two-artifact configuration, and Workspace N with a three-artifact configuration. The extended model can be extended further to contain even more intermediate repositories, if needed. A similar approach was described by Hall et al [6].

2.3 Case Study Validity

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.

2.3.1 Case Selection

The ChipSoft case is relevant to our research project since ChipSoft works in a problem domain with specific characteristics, in this case health-care information systems. Examining this case allows us to see the release, delivery, and deployment activities in a real-life company that is significantly different from our previous cases in other domains [7, 8]. 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 [7] and Planon [8], which allows us to compare the type and number of customers to a company's release, delivery, and deployment practices. Finally, the software produced by ChipSoft provides customers with a large amount of freedom to customize their installed software. The interaction between these customizations on the one hand and installing, upgrading, and configuring on the other hand is also relevant to our research.

2.3.2 Information Gathering

During the case study, we collected information to answer the four research questions and achieve the two research goals. This information was gathered through the following means:

- *Interviews*
  We interviewed the employees responsible for the development, release, delivery, deployment, and usage of the software.

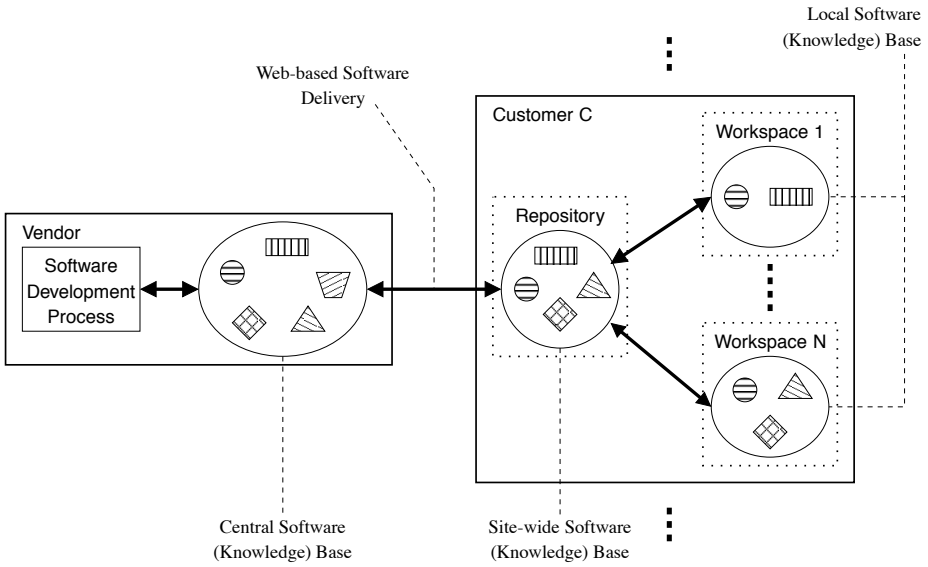

Figure 2: Extended software deliver model for ChipSoft.

- *Studying the software*
  We examined the architecture, organization, and characteristics of the software.
- *Document study*
  We studied various documents, for instance, on development, release, and usage.
- *Direct observations*
  The software product and several development and release support tools were demonstrated to us.

These means for gathering case study facts are largely of a qualitative nature and do not allow any statistical analysis. This is not a problem, however, since this case study is mostly of an exploratory nature [15].

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 following types of personnel found at customer sites could provide useful information, they were not approached due to practical and business-related considerations.

- System administrators
- End users

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.

Table 1: Milestones in ChipSoft's history.

1986	Medical practice automation
1988	OR & IC automation
1990	Departmental systems
1994	Hospital information system
2001	Electronic care information system

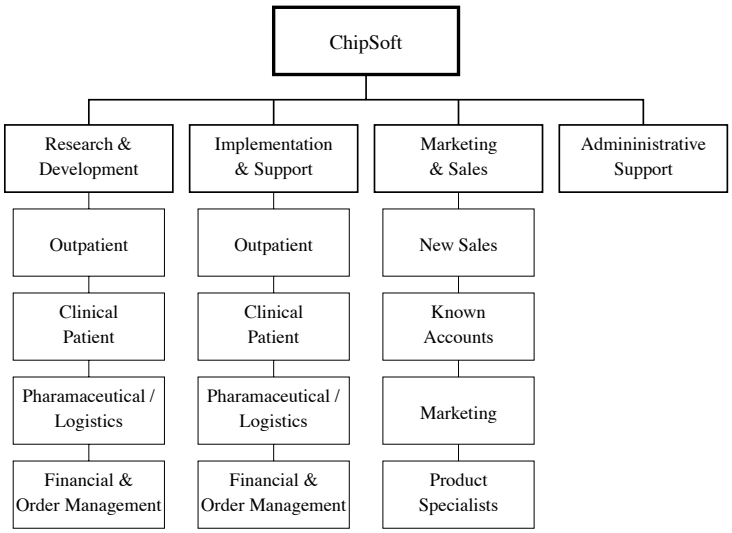

Figure 3: Organizational overview of ChipSoft with its four departments.

3 ChipSoft

ChipSoft is a Dutch medium-size company that provides comprehensive ICT solutions in the medical domain. The company's main business activities are the production and sale of medical information systems, the customization of these products 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. Table 1 shows the most important milestones in ChipSoft's history. ChipSoft started with producing applications for the automation of the primary care process(es) of medical specialists. Over the following years, ChipSoft extended and 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). [1]

As shown in Figure 3, ChipSoft currently consists of four departments: Research & Development, Implementation & Support, Marketing & Sales, and Administrative Support. The core business activities are spread over the first three departments, with the last department providing basic secretarial and administrative support. Table 2 shows how the ChipSoft's 100 employees are distributed over the four departments. ChipSoft is also geographically split with its main site in Amsterdam and a smaller site in Drachten, in the north of the Netherlands. Most employees (approx. 86) work at the main site while fourteen software engineers of the Research & Development department work at the site in Drachten.

The Research & Development department consists of software engineers, and is responsible for the

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

Table 2: Approximate number of employees per department.

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

Table 3: Category and module list.

Outpatient and clinical care logistics and care registration	
CS-Patient	CS-Diary
CS-Episode	CS-Admission
CS-Department	CS-OR
CS-Preop	Anesthesia Record
CS-Sterile	CS-A&E
CS-X-Ray	CS-Archive
Electronic patient record (EPR)	
CS-Info	CS-Digital Record Registration
CS-Medication	CS-Document
CS-Order	CS-MultiMedia
Financial settlement and management information	
CS-Invoice	CS-Ambulant
CS-Overview Generator	CS-Datawarehouse

development and maintenance of the core application functionality. The Implementation & Support department, in contrast, consists of consultants, and is responsible for the implementation of the application at specific customers, including customer-specific customizations. Implementation & Support is also responsible for providing tutorials on the use of ChipSoft products. Both departments are subdivided to deal with the four main application subdomains: Outpatient, Clinical Patient, Pharmaceutical/Logistics, and Financial & Order Management.

4 ChipSoft Products

4.1 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 that are grouped in three categories. These three categories and their respective modules are shown in Table 3. 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 is therefore more accurately described as a software product family than as a single software product.

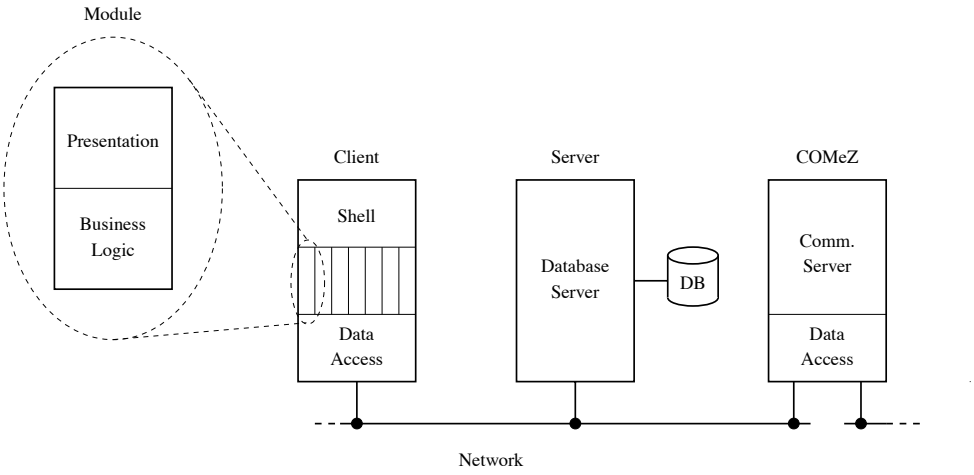

Figure 4: The CS-ECIS architecture consist of clients and a server, where the client is divided into a shell application with a variable number of extension/implementation modules. The communication server COMeZ is a special kind of database client.

As shown in Figure 4, 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.

4.2 COMeZ

To enable the integration of CS-ECIS with other (third-party) systems used in a hospital, for example X-ray machines, ChipSoft provides a communication server, called COMeZ ("Communication Easy"), also shown in Figure 4. This communication server can be considered a special kind of database client that runs on a separate computer and interacts with both the database server and third-party hard- and software. To improve overall efficiency of the system, COMeZ can also provide a backing store to cache the data generated by these third-party systems.

To enable the integration, COMeZ performs protocol and format conversions on all protocol layers, from the physical layer up to the application layer. These conversions are called bindings.[2] COMeZ supports, for instance, bindings for the HL7 (health-care level 7) information exchange protocol, but also for SOAP-based services. COMeZ uses a plugin architecture to ensure it can easily be extended with new bindings. It currently support thirty bindings, which are implemented in six plugins, and is built from approximately 600,000 lines of Delphi code.

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

[2] In Dutch: koppelingen

- User profiles
- GUI modifications/additions
  - Layout and contents of screens
  - Layout and contents of reports
- Functionality modifications/additions
  - User-defined fields
  - User-defined expressions
  - User-defined actions
- Customer-specific modules

Since different customers can buy a different set of modules, the modules bought by a customer need to activated before use. Recall that the *complete* set of modules of CS-ECIS is always distributed to a customer. This variability is obviously only changeable by ChipSoft itself. 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. The default GUI is stored in the application, and changes to it are stored in the database as XML-encoded data.

When the information that is stored, computed, or shown by CS-ECIS, is incomplete or not usable by a customer by default, 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 (user specified) value can be computed. If that is not enough, the end-user can also add new new fields to existing tables. 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 Research & Development department. This module then becomes part of the main code base, and is thus distributed to all customers, but is activated only for that particular customer. The customer-specific module can obviously also be activated for other customers if the module proves to be more generally applicable. Note that CS-ECIS does not support module variants, that is, interchangeable modules that provide similar functionality.

COMeZ supports variability in its bindings (i.e., conversions) in ways similar to CS-ECIS:

- Activation of (types of) bindings
- User-defined expressions
- User-defined actions

- Addition of new plugins (DLLs)

Similar to CS-ECIS, a customer can buy one or more (types of) bindings and have them activated, but all plugins are always delivered to the customer. Note that a single (type of) binding can support multiple binding instances, that is, bindings to different third-party soft- and hardware. The largest amount of customizability of the conversion comes from the user-defined expressions and actions. The expressions and actions enable end users to specify how a protocol or format should be converted to another. If the expressions and action functionality is not powerful enough, the Research & Development department can develop new a binding or plugin. COMeZ can then be extended by simply installing the DLL that implements the new binding and restarting the application.

4.4 Platform

All ChipSoft software is targeted toward the operating systems sold by Microsoft. Until recently, for instance, ChipSoft still built software products for the MS-DOS operating system. For the current versions of CS-ECIS (i.e., the client) and COMeZ, ChipSoft requires the Windows 98 operating system or later. For the database server, a version of Windows 2000 or later is needed. 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. A setup that is frequently found in hospitals uses 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.

5 Release, Delivery, and Deployment

5.1 Release Management

5.1.1 Releases, Service Packs, and Hotfixes

ChipSoft uses three types of software distributions to provide customers with new versions of their software: releases, service packs (SPs), and hotfixes (HFs).

Twice a year (i.e., in spring and in autumn), ChipSoft creates a new *release* of the CS-ECIS software (incl. COMeZ). To receive support, customers are (contractually) allowed to be at most two releases behind the current release. Between releases, ChipSoft sometimes provides a service pack. A *service pack* is created, whenever the software has significantly changed, the changes are required immediately by customers, and the next release is too far away. 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 hotfixes and service packs, including the release on which it is based. ChipSoft supports a separate release, called a feature pack, that contains experimental features implemented for specific organizations (see also 6.2.1).

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. Even though the 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 ("a showstopper"), 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*. This practice avoids 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.

5.1.2 Distribution Identification

ChipSoft uses two ways to refer to its software distributions: 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 a release name (the release 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 a release or service pack name (the release or service pack 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. 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".

5.2 Implementation and Support

After buying one or more modules of CS-ECIS, ChipSoft offers its customer (i.e., a hospital) extensive support for implementing the product(s) in their business processes. The responsibility for this implementation support is placed on the Implementation & Support department of ChipSoft. The department is basically responsible for everything between the contract signing and the application "going live." Afterward they service as the contact point for that particular customer.

In more concrete terms, this implementation support consists of:

- (Help with the) Initial installation of the test setup
- Elicitation of the detailed requirements
- Planning of implementation
- Assistance with the configuration of the software
- Mediating with the Research & Development department

- Knowledge transfer (i.e., training) and general assistance

The elicitation of detailed requirements is an important aspect of the implementation phase since the license contract contains only a global description of the requirements of the customer. The detailed requirements are needed to determine how CS-ECIS need to be configured and how long this is going to take. An important reason for the amount of support given by the Implementation & Support department, is the amount of configurability provided by CS-ECIS. Implementation & Support enables a customer to configure CS-ECIS to be effective and efficient in its particular computing environment.

If a customer desires a customization that cannot be achieved through configuration, some additional programming needs to be done. This happens, for instance, when CS-ECIS needs to integrate with previously unknown, third-party hard- and software. The actual software development is, in such as case, delegated to Research & Development department with the Implementation & Support department performing a mediating role. The Implementation & Support department also provides seminars for application managers (i.e., knowledge transfer). These managers can then organize their own seminars, tailor-made for their particular end-users. An unrelated responsibility of the Implementation & Support department is to test new releases before they are distributed.

5.3 Deployment

ChipSoft offers no support for the deployment (installation) of CS-ECIS software inside a hospital. Customers (i.e., the hospitals) are themselves responsible for the internal deployment CS-ECIS, and are expected to use, for instance, COTS products. The main reason for not providing any deployment 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 the deployment CS-ECIS.

6 Development and Maintenance

6.1 Development Support

ChipSoft develops CS-ECIS and COMeZ using the Delphi programming language, and although most of their source code is in Delphi, some other programming languages are also used. For instance, the graphical user interface (GUI) customizations are recorded in an XML-based format, and to process these GUI customizations, XML stylesheets (i.e., transformation specifications) are used. Furthermore, the client application has recently started to use queries in the SQL language stored in the database server.

The main programming tasks of the software developers, such as editing, compiling, and debugging, are supported by the Delphi integrated development environment (IDE). To create global builds for testing and release, ChipSoft uses a dedicated build server that runs the FinalBuilder build tool from Atozed. [3] For version management, ChipSoft uses Microsoft's Visual SourceSafe (VSS). To administrate defect information, ChipSoft uses the open source defect tracker Mantis. [4] This system is for internal use only. To record defect reports and enhancement requests from customers, a separate, locally developed, customer relationship management (CRM) system is used. This system also records customer information, including information about contracts.

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 images, for instance, from X-ray machines, ChipSoft uses PACS (picture archiving and communication system) software. This software is provided by the hardware vendor. Apart from XML stylesheets, ChipSoft also uses some Microsoft code to read and write XML data. Finally, to create reports from the database for management purposes, the report builder software from Digital-Metaphors[5] is used.

[3] http://www.atozed.com/
[4] http://www.mantisbt.org/
[5] http://www.digital-metaphors.com/

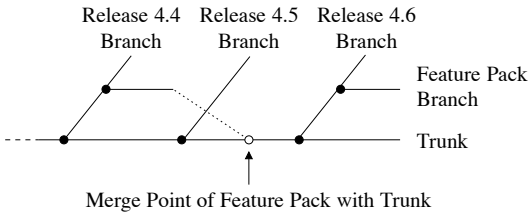

Figure 5: Version Tree.

6.2 Development Process

6.2.1 Version Management

In its VSS source code repositories, ChipSoft uses mostly pessimistic concurrency control (i.e., locking). Locking source files is not perceived as a bottleneck, however, since the development work is distributed in disjoint subsets over the developers anyway. To enable concurrent development and maintenance, ChipSoft uses the simple branching scheme shown in Figure 5. The upcoming release is stored in the trunk of the repository and the most recent releases are stored in branches. Older releases are stored as branches in a separate backup repository.

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 branch, called a feature pack, is created. 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. As a consequence, when multiple special features are developed concurrently, the feature pack will contain the results from all development efforts. As an example, Figure 5 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 COMeZ and their 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 or COMeZ. As a consequence, consistency between artifacts can be maintained using their version management tool (i.e., VSS). There is currently no strong integration between ChipSoft's version management system (VSS) 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 is fixed.

6.2.2 Product Structure

The (internal) structure of CS-ECIS and COMeZ executable is described using Delphi projects. The Delphi IDE and FinalBuilder use these project descriptions (plus some extra configuration information) to build the executables and DLL files. ChipSoft uses one Delphi project per executable or DLL, but projects can have subprojects when this is more convenient. Every project can consist of multiple files.

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 and COMeZ. As a consequence, these internal dependencies, for instance caused by intermodule 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 on the expression language implemented by a particular release of the client and COMeZ. This dependency problem is dealt with by keeping the expression language backward compatible.

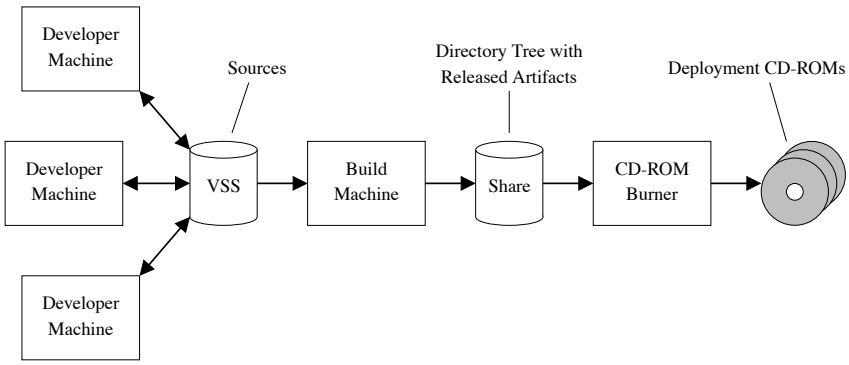

Figure 6: Development and release workflow in ChipSoft.

6.2.3 Build and Release Process

The global development and release flow is shown in Figure 6. Developers work at their workstations and commit new and modified artifacts to the central VSS repository. ChipSoft uses a single central build server for global builds. This server provides a clean build environment that does not depend on specific workspace configurations of developers. The build server compiles the sources 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.[6]

The global build server uses the artifacts stored in the repository. A global build is started manually, usually twice per day. A global build is specifically requested whenever the interface of a DLL has changed and an integration test is needed. To avoid having to rebuild everything themselves, software developers can download the latest DLLs from the build server for use in their private IDEs.

6.2.4 Testing

ChipSoft uses no formal techniques to verify the correctness of a their software. Instead, correctness is maintained only through testing. The build server performs a simple integration test by providing a clean build environment. The use of the build server thereby provides 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 and the use-case tests are described in a manual. Since these tests require a significant amount of effort (about two weeks), ChipSoft only (formally) tests releases and relies on its software engineers to test their modifications thoroughly themselves. ChipSoft does not use any automatic unit, smoke, or integration test.

Since various hospitals can have a very different setups, ChipSoft cannot test its new releases in each and every setup. However, since the software is mission critical, hospitals cannot simply install the software and hope for the best. Instead, to deal with this compatibility testing 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.

6.2.5 Internal Communication

To avoid problems with major changes to the software, such as adding a new module, ChipSoft uses documented procedures for these changes, described on the internal website. An example of such a procedure, the addition of a module, is shown in Table 4. For the most part, however, the source code and more experienced programmers are considered the main sources of information. ChipSoft organizes an internal seminar twice a year to improve the overall knowledge of its programmers of the software.

[6] A more accurate name would be a *transport* CD-ROM, instead of an installation CD-ROM.

Table 4: Adding a new module to CS-ECIS

1	Create a subproject for the business logic DLL.
2	Create a subproject for the presentation (GUI) DLL.
3	Add module to module table in the database.
3	Add module to the data dictionary in the database.
4	Add the project to the global project.

7 Discussion

In Sections 3–6 of this report, we have "describe[d] the current release, delivery, and deployment activities of ChipSoft" to achieve research goal G1. In order to achieve research goal G2, we "relate these activities to the activities of the Deliver model" in this section. We perform this analysis by answering the related research questions Q3 and Q4:

Q3 How do these processes and techniques relate to the processes and techniques proposed by the Deliver model, such as the ISKB?

Q4 What processes and techniques can be improved or extended, both at ChipSoft and in the Deliver model?

7.1 Relating the Deliver and ChipSoft models

To answer question Q3, we focus on the five main characteristics of the Deliver model, and use these to compare the Deliver model to the current ChipSoft approach.

7.1.1 (Highly) Modularized Software

The first aspect of the Deliver model is a strong focus on modularized software. This (strong) focus refers to the use of the module concept in all phases of the software life cycle, such as, design, implementation, release, delivery, and deployment. A strong focus on modularized software is useful for an organization for a number of reasons. To start with, modularization enhances the potential for software reuse. It simplifies providing incremental updates for defect fixes and enhancements, and is required for the late binding of (third-party) software, which facilitates product variability and extensibility. Finally, modularization also helps in resource-constrained environments and to protect trade secrets.

At ChipSoft, modularization is used mostly as a design and marketing technique. For instance, all modules of the CS-ECIS software are always present in an installation. COMeZ provides some extensibility by allowing new plugins to be added after the initial deployment. Without a need to support (significantly) different products, pushing for more modularization, for example in the runtime support, is not particularly warranted at ChipSoft, particularly, since the applications are not run in a resource-constrained environment and trade-secret protection is not needed. ChipSoft currently supports incremental updates at the DLL level, but these updates are unrelated to the product modularization.

7.1.2 (Highly) Configurable Software

The second aspect of the Deliver model is a strong focus on software variability [14]. This aspect refers to a focus on configurable modules, variable configurations of modules, and variable architectures. Examples of the result of this focus are product families, product lines, and product populations. This strong focus on configurable software is useful for an organization since it increases the potential for reuse. Furthermore, variability can improve user satisfaction and enlarge the (potential) customer base.

At ChipSoft, variability is achieved at the product feature level through its use of a configurable and extensible GUI, database, and bindings. ChipSoft does not maintain variability at design, implementation, or release time, nor at the architectural or management level. To support internationalization at some point

in the future, all GUI texts have recently been externalized (i.e., stored in a resource file) to allow them to be easily changed. The current approach to variability seems adequate, and without the desire to support (significantly) different products, pushing for more variability support at ChipSoft is not particularly warranted.

7.1.3 Incremental Updates

The third aspect of the Deliver model is support for incremental updates. Support for incremental updates means providing the ability to use (i.e., communicate, store, and apply) only the difference between an old and a new configuration instead of the complete new configuration. To simplify the conceptual model, incremental updates should be managed at the module level, and this feature usually requires support for independent versioning of modules, instead of versioning of the whole application. Incremental updates are useful for an organization since they provide a short release cycle, but efficient (and low risk) use of incremental updates does require automated support for computing the difference between configurations.

ChipSoft supports incremental updates through its use of hotfixes, but these hotfixes are only low level and do not refer to specific modules. These hotfixes are only short lived, that is, valid only until the next service pack or release arrives, and used simply as low cost way to transform the installed application to a new version. ChipSoft currently has no automated support for determining what should be part of the hotfix, and relies on the programmer to keep track of which files have changed. Incremental updates could also be used for releases and service packs, but if most files have changed between versions, the gain of using incremental updates would be small.

7.1.4 Network-based Software Delivery and Deployment

The fourth aspect of the Deliver model is network-based software delivery and deployment. Network-based software delivery and deployment refers to comprehensive use of computer networks, such as the Internet, to deliver software as either full packages or incremental updates, preferably using protocols that take versioning information into account. Network-based software delivery and deployment is useful since it lowers the overhead of providing new versions and shortens the release phase of a product.

ChipSoft currently uses network-based software delivery only for its hotfixes where a short release cycle is needed. The protocols used are, however, only generic WWW protocols that are not specific to incremental updates. Since ChipSoft does not support deployment, network-based software deployment is not available. Network-based software delivery of releases and service pack would be possible for ChipSoft if the appropriate security mechanisms are put in place. The information gathered in this case study suggests, however, that the integration of new releases, and not the software delivery, is the bottleneck in the whole release, distribution, and deployment cycle.

7.1.5 Intelligent Software Knowledge Base

The fifth aspect of the Deliver model is the use of an intelligent software knowledge base. Using an intelligent software knowledge base enables the gathering and combining of information about the software and its processes, which simplifies the management of these processes. The ISKB requires a formalized model of the software product and its processes. The intelligent software knowledge base is useful for an organization since it automates and simplifies the management of software processes, thereby lowering its overhead and improving the quality of the software product. The ISKB can be used both to drive the development, release, and deployment process and to verify product quality and progress.

ChipSoft currently does not use anything resembling an intelligent software knowledge base. Only a limited amount of information (e.g., build structure and defect tracking) is stored and that 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. Implementing an intelligent software knowledge base at ChipSoft would unfortunately be difficult and have a high impact on the organization at present since the required information and the processes needed to create it are currently not present. Furthermore, the business processes of ChipSoft are not complex enough to warrant an intelligent software knowledge base.

7.2 Possible Improvements and Extensions

In this section, we discus possible improvements to ChipSoft activities and to the Deliver model. Since ICT management in the medical field is very conservative nature, preferring stability over state-of-the-art features, ChipSoft desires to keep its product architecture stable for the next five years. We therefore discuss low-impact and high-impact improvements to ChipSoft separately. We finish this section with a discussion of improvements to the Deliver model.

7.2.1 Low-impact Improvements

The low-impact improvements extends current practices, and can be summarized as follows:

- *Extend the use of network-based software delivery.*
  The current development, release, and delivery approach for hotfixes is also suitable for network-based delivery of releases and service packs.
- *Stronger integration of CRM, defect tracking, and version management.*
  A previous case study at Exact Software [7] has shown that this integration can significantly improve insight in the software product and its development process, promoting their quality.
- *Increase the amount of (automated) testing.*
  A surprising finding was the limited amount of structured (e.g., automated) software testing being done at ChipSoft. An obvious suggestion would be to apply the ideas of test-driven software development, for instance, by introducing code reviews or unit testing.

7.2.2 High-impact Improvements

The high-impact improvements require significant changes to the current architecture and business case. They can be summarized as follows:

- *Introduce explicit representation of knowledge in the development and release processes.*
  For instance, to automatically check quality aspects, such as consistency of full packages and incremental updates. This change requires clear goals on what quality aspects to improve.
- *Introduce support for network-based deployment.*
  The logical conclusion of network-based software distributions is network-based software deployment. Such a deployment tool can take ChipSoft-specific knowledge into account, potentially simplifying deployment process for the customer.
- *Introduce Customer Site Information Gathering.*
  Supporting CS-ECIS with custom tools, such as deployment tools, enables the gathering of information actual customer configurations. This information can provide insight into what configurations are actually used by customers, including interaction with third-party software.
- *Improve Testing of Software Configurations.*
  Insight into the actual configurations used also enables more effective use of testing resources. This idea can be extended, for instance, with the on-line dependency analyzer (OA) of Microsoft Research [3]. This analyzer would show ChipSoft customers what functionality to check before rolling out a new release.

7.2.3 Improvements to the Deliver Model

When we consider improvements to the Deliver model, in contrast, the case study provides, unfortunately, little to learn. There are two reasons for this lack. The first reason is that the deliver and deployment aspect of Deliver has not been applicable to ChipSoft. Since ChipSoft is not involved in the deployment of their software at their customers, very little could be learned in this area. As a consequence, we have, for instance, not been able to evaluate the extended software delivery model we proposed in Figure 2. The

second reason is that, apart from build tools, ChipSoft does not (explicitly) manage its software knowledge. ChipSoft has little need for formalized software knowledge since it manages to keep its approach to version and release management simple but effective. As a consequence, this case study does raise questions about the general applicability of the Deliver model and the intelligent software knowledge base.

8 Conclusions

This report describes the results of a case study we performed at the Dutch software vendor ChipSoft. This case study had two goals. The first goal was to describe the development, release, delivery, and deployment process of ChipSoft. The second goal was to compare these processes with the processes suggested by the Deliver model. 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. The case study does provide the possibility of comparing this case with the Exact Software [7] and Planon [8] 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] Ed Bailey. *Maximum RPM*. 1997.

[2] ChipSoft. `http://www.chipsoft.nl/`.

[3] John Dunagan, Roussi Roussev, Brad Daniels, Aaron Johnson, Chad Verbowski, and Yi-Min 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.

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

[5] Exact Software. `http://www.exact.nl/`.

[6] Richard S. Hall, Dennis Heimbigner, and Alexander L. Wolf. A cooperative approach to support software deployment using the software dock. In *Proceedings of the 21st International Conference on Software Engineering*, pages 174–183, Los Angeles, CA, USA, May 1999.

[7] Remy Jansen, Gerco Ballintijn, and Sjaak Brinkkemper. Software release and deployment at Exact: A case study report. Technical Report SEN-E0414, Centre of Mathematics and Computer Science (CWI), Amsterdam, The Netherlands, September 2004.

[8] Slinger Jansen, Gerco Ballintijn, and Sjaak Brinkkemper. Release and deployment at Planon: A case study. Technical Report SEN-E0504, Centre of Mathematics and Computer Science (CWI), Amsterdam, The Netherlands, March 2005.

[9] Slinger Jansen, Sjaak Brinkkemper, and Gerco 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, March 2005.

[10] Paul Klint. Intelligent software knowledge management and delivery, NWO Jacquard grant proposal. `http://www.cwi.nl/themes/sen1/twiki/pub/Deliver/Publications/proposal.pdf`, June 2002.

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

[12] Planon. `http://www.planon.nl/`.

[13] RPM website. `http://www.rpm.org/`.

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

[15] Robert 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.