# Variability Issues in the Evolution of Information System Ecosystems

Hendrik Brummermann
Hochschul-Information-System GmbH
Goseriede 9
30159 Hannover, Germany
brummermann@sse.uni-hildesheim.de

Markus Keunecke
University Hildesheim
Institut fuer Informatik
Marienburger Platz 2
31141 Hildesheim, Germany
keunecke@sse.uni-hildesheim.de

Klaus Schmid
University Hildesheim
Institut fuer Informatik
Marienburger Platz 2
31141 Hildesheim, Germany
schmid@sse.uni-hildesheim.de

## ABSTRACT

In a software ecosystem with open variability customers create their own products based on a reuse infrastructure provided by a development company. While an open approach has many benefits, it brings along a number of specific issues, especially related to evolution. In this problem statement we discuss some of the issues that arise in merging local variabilities with evolved versions of the reuse infrastructure of the development organization. In our discussion we focus on information systems, inspired by the situation of a specific company.

## Categories and Subject Descriptors

D.2.7 [**Distribution, Maintenance, and Enhancement**]; D.2.13 [**Reusable Software**]: Reuse Models; H.3.5 [**Online Information Services**]: Web-based services

## General Terms

Design, Management, Experimentation

## Keywords

Variability Modeling, Evolution, Software Product Lines, Software Ecosystem, HIS

## 1. INTRODUCTION

Traditionally product line engineering focuses on the development of software within a single company [2, 7]. More recently, however, the relationship of software ecosystems with product line engineering was pointed out [1]. This situation is more complex as software can be developed by multiple organizations and this happens to some extent without explicit synchronization. In particular, a customizable software platform may be built by one organization, while another organization builds software on top of this [5].

Such a development approach, which relies on open variability is particularly common in the information systems area. The approach, however, is accompanied by an additional level of complexity, which we will address in this paper. Based on a case study of a company that supports such a development model, the specific issues of variability management that are derived from such a development situation are shown. More precisely we will discuss issues of variability management that are connected to open variability, updating and evolution in such a situation, and information systems variability.

The remainder of this paper is structured as follows: Section 2 describes the context. In the following section we discuss the issues that arise in an open software ecosystem before we conclude the paper in section 4.

## 2. STUDY CONTEXT

Many of the problems we will discuss in this paper are of a generic nature, however, our discussion is inspired by a specific company context: Hochschul-Informations System GmbH. In this section, we will describe this context in order for the reader to get a better understanding of the following section. Thus, we will introduce the company, the software it is developing and its approach towards interacting with customers.

### 2.1 Company HIS

HIS Hochschul-Informations-System GmbH in Hanover is market leader for university management software in Germany. It was founded in 1969 as a non profit company. Its main product was for many years a desktop-based system which consisted actually of a set of different subsystems, that could be installed independently and used at the customer site. The various subsystems targeted different problem areas and work contexts like accounting, student management, lectures, etc.

In 2007 the development of a new software generation called HISinOne was started. As opposed to the earlier generation, HISinOne was conceived as a web based system and organized around business processes which may cover multiple different work contexts. It provides a consistent user interface which can also be customized for the different roles of users such as administration, lecturer and students. Figure 1 gives a first impression of the University Management System under discussion.

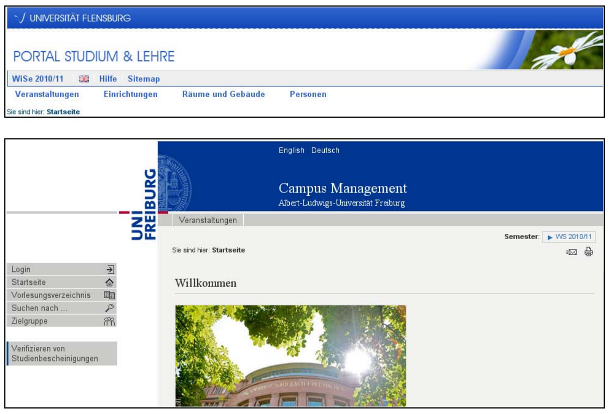

**Figure 1: Customizations at the University of Flensburg and the University of Freiburg**

## 2.2 Development Model

HISinOne installations are more than yet another system in a system landscape. Actually HISinOne is at the core of its own software ecosystem. While the central capabilities are developed by HIS, any university is allowed to adapt and extend HISinOne independently. Often customers also contract the solution department of HIS and ask for specific modifications and extensions, but even in this case further extensions are often done by the customers. Third party service providers and independent contractors offer solutions based on HISinOne, thus augmenting the software ecosystem with further capabilities. At the same time competitors provide alternatives for specific modules of HISinOne capabilities with which HISinOne must be able to cooperate in customer installations.

HISinOne is developed as a highly customizable system. This is due to the high diversity of usage contexts of the system. This diversity stems from multiple sources: Germany is divided in 16 regional states which a high degree of autonomy in terms of regulating their university system. Further, different types of universities and schools exist (e.g., University vs. University of Applied Sciences vs. Art Academy). All these differences lead to different requirements. Each university can define its own organizational model within certain bounds and also make its own decisions regarding study programs, lecturers, and so forth. This demands a significant level of customization for each university and entails the need for high customizability of the HISinOne products. However, customization does not stop at the site level, even individual users may customize their way of using HISinOne.

Many different forms of customization are thus supported, relating to look-and-feel, corporate design, supported business processes and information managed, and so forth. Figure 1 shows two screenshots from the Universities of Flensburg and Freiburg. Both universities have adapted the software to their corporate design and business processes on their own. The University Flensburg uses objects such as lectures and institutions as the top level navigation entries. But the University Freiburg has organized the navigation menu by tasks such as “Searching”.

While HIS provides support and consulting, customers are free to do arbitrary changes on their own. Therefore HIS does not know about all customer changes. But, it is very important that updates provided by HIS do not break existing adaptations at the customer side — or in case this is not possible, these problems are minimized.

HISinOne is a special case of a software ecosystem because customers and independent contractors develop customizations and extensions. Ommering discusses product populations as a reuse concept between related product families that are developed by different suborganizations in [8]. Although the approach of HIS is similar, it faces new challenges because the development is split across several organizations. An approach to partial instantiation of software product lines with updates, feedback and cross-merges is described by Krueger in [4]. Krueger sees cross merges between products mostly as an interim solution before the changes to common assets can be fed back to the reuse infrastructure. For HIS, however, it is a normal situation that there is no complete feedback of customer modifications. This means that there is no global, coordinated view on the complete variability and decision models in the product line. While Bosch describes what software ecosystems are in [1], this paper discuses specific issues that arise in a specific but common case.

| Segment | Functional Area |
|---|---|
| CS | Identity Management (PSV) |
| CS | Community (COM) |
| CS | Business Intelligence (BIA) |
| CM | Applicant Management (APP) |
| CM | Students Management(STU) |
| CM | Lecture & Tests (EXA) |
| CM | Alumni Management (ALU) |
| RM | Financial Accounting (FIA) |
| RM | Financial Management (FIM) |
| RM | Human Resources (HRM) |
| RT | Research Management (YRM) |
| RT | Technology Transfer (YRT) |

**Table 1: Functionalities covered by HISinOne**

## 2.3 Capabilities of HISinOne

HISinOne covers many functionalities important to universities as shown in Table 1. They are divided into four segments:

- Core Segment (CS) provides common services such as identity management and business intelligence.
- Campus Management (CM) deals with functionalities specific to universities such as management of students and lectures.
- Resource Management (RM) deals with resource planning, such as accounting and human resources.
- Research Management (RT) deals with the management of research projects and technology transfer.

HISinOne aims to cover the whole range of functionality relevant to management within a university, as Table 1 shows. We will illustrate this below by a brief discussion of various functionalities.

Applications from prospective students are handled online by Applicant Management (APP). This also supports applicant selection based on locally relevant criteria. Once the applicant is admitted he becomes a student and all management of fees, documents, grades is handled as part of Students Management (STU). Further Identity Management (PSV) is relevant to assign a unique ID to the student. PSV also provides an identity for other objects such as rooms and organizational units that are used in many different contexts. As part of a study program students need to take exams to prove their study success. Management of this is handled by the Lecture & Tests functionality (EXA). This includes scheduling the tests, collecting the results and grade calculation and documentation.

Depending on regional regulations students have to pay study fees. This is also handled via Students Management and the results are forwarded to the university's Financial Accounting (FIA). Fees and other income affect the university budget, which is handled in the Financial Management domain (FIM). More general information such as the expected and actual numbers of applicants per course of study are collected and aggregated in the Business Intelligence Domain (BIA). People employed by the University like lecturers or administrative staff are handled by Human Resource Management (HRM).

HISinOne also addresses marketing towards alumni, which is handled by the Alumni functionality (ALU). This set of functionality also supports fund raising and public relations. It is closely related to an online Community (COM) which allows alumni to stay in contact with their fellow students.

An important part of university work is research and technology transfer. The functional areas of Research (YRM) and Technology Transfer (YRT) cover this.

None of the above areas represents a single, mandatory functionality, but all of them contain major variability. These variabilities are resolved in the specific implementations of each university. When new versions of HISinOne are released, it is the responsibilty of the customers to merge them into their local products.

## 3. PROBLEMS

In this section, we will focus on a discussion of the various kinds of variability that are relevant to information systems and the kind of issues that arise with them from the combination of open variability and product line evolution. More precisely we will structure our discussion based on the different types of variability in information systems, going from aspects that are close to the user towards more technical aspects. The different addressed areas are the user interface, business rules, business processes and data. For each of these areas we will then discuss different problems that arise with respect to the evolution of variability and illustrate them, if needed, using examples from the HISinOne-system.

### 3.1 User Interface

Customers can modify the user interface in HISinOne on two levels: first they can make explicit code modifications or they can modify configuration settings. Both types of changes must be respected in case an update of the central product line infrastructure is rolled out to the customer site. In this section, we will mostly focus on the first kind of modification, as the second will be addressed also in the section regarding data.

Customers adjust the user interface in order to allow efficient working according to their local requirements. This is done by hiding unused functions and input fields to reduce the risk of errors and number of support calls. In case many input fields are hidden, the remaining fields may be spread across several almost empty pages. Moving them to a small number of pages improves usability by providing an overview of the important data at a glance. Further, in case data from paper forms is entered into the system, the input speed can be increased by using the same order of input fields in the system as on the paper forms.

Many customers also want to apply their corporate design to the university management system to have a coherent look-and-feel throughout the various information systems and also some parts of the system are visible to the outside.

These improvements of the user interface can incur significant effort. Ideally, updates to the user interface would preserve these kinds of changes, as far as possible. Of course, in certain situations this is not easily possible, e.g., if new fields are added due to new data that must be gathered according to changed laws, this may lead to certain pages growing again and thus to a need to split pages. For example in 2010 a central German registration system for applicants was introduced by law. As a result the web forms at the universities have to provide an additional input field for the central registration number.

A way of describing user interfaces — and their variability — which allows for easy and context sensitive merging is needed. It has to take into account different kinds of changes: The composition of masks (web pages) may be adjusted by adding, deleting or modifying fields and texts. Additional functionality and specific ways of entering data may be added to fields. Finally changes to the look-and-feel (e.g., corporate design) need to be specified.

As this leads to very complicated interactions of adaptions, we see a problem with existing approaches like feature modeling [3] or decision modeling [6] in terms of representing this, even beyond the issues of evolution and open variability.

### 3.2 Business Rules

Customers may change business rules, but changes that are destructive to the system or violate common law should be prevented. Updates of the HISinOne-platform that are rolled out to the customers must preserve those changes where applicable, but remove them when new regulations must be enforced.

Business rules within HISinOne in the base configuration can be categorized in two ways: first they can be categorized as validation rules vs. computation rules. This means, sometimes they are used to enforce certain standards, while sometimes they are used to derive data values. Secondly they can be categorized based on whether they are required by all installations (although they have a different form in them). The format of registration numbers for students provides a simple example of a business rule for validation. These may be added as validation rules. While customers may define the specific format, they are not allowed to remove the rule altogether, as this field is required by law.

In addition to the mandatory rules that are based on law or technical requirements, HIS provides a set of default rules useful for most universities. Their existence can be considered a variability. For example the lecture number is marked as mandatory in the default rule set because many customers use it in their business processes. However, as it is not required by the system this rule may be removed by the customer. So from the point of view of variability it actually provides a point of variability. In case a customer removes it any update should not reintroduce any such rule. It should be noted that we need to differentiate whether the existence of a specific business rule is mandatory or the form of a business rule is mandatory. In most cases we are actually concerned with a situation where only the existence is mandatory.

In total we can differentiate the following cases when updating a business rule:

- the category of a business rule remains the same (i.e., either mandatory or variable). In this case any adaptation made by the customer should be preserved. This also means that any functionality required for this adaptation should be preserved as well.

- if the category changes from variable to mandatory and the customer deselected the business rule, the update must enforce the use of a rule (although the customer may still determine the specific form).

- if the category changes from mandatory to optional, this is no problem as the customer already has a rule and it is still valid. However, in the future the customer may remove the rule.

## 3.3 Business Processes

In information systems in general and university management systems in particular online business process models play an important role. Process models reflect business processes from the real world in an information system. Examples for business processes are handling requests for leave or planning lectures.

Business process models are commonly described by defining activities such as "Grant application for leave" and transitions. Activities are assigned to actors. Transitions link activities and may have conditions attached. An activity consists of all steps one person does within a business process before forwarding the work to the next actor. The system may support these steps by providing one or more business functions.

The development organization provides a set of reference business process models and activities that customers may use as is or adapt to their needs. New versions of the reuse infrastructure may contain changes of business process models and activities as well, leading to the need to look at situations concerning both activities and process models.

Adding new activities is the simple case because no conflicts can occur. The development company may want to delete outdated activities. As it is desirable not to break customer specific process models, those types are not deleted but flagged as deprecated. If customers want to delete unused activities, a similar approach is useful because future process models provided by the development organization may use them. Activities may be evolved in the product line infrastructure and those changes shall be applied to the customer installations. But customers may want to modify activities as well, for example an activity "inform students" may be implemented by putting up a piece of paper on the blackboard instead of sending emails. As long as an activity is only edited by one organization, those changes can be applied. If both the customer and the development organization modify the same activities, manual merging may be necessary.

Business process models can be added by both the development organization and the customers without conflicts. The development organization must not delete process models because there may be running process instances at the customers' sites. They need to be flagged as deprecated in order to allow finishing of running instances. Customers may delete process models if and only if there are no running instances. If the process model has been in production, a similar deprecation mechanism is needed to prevent new process instances from being started. In rare situations it may be necessary to migrate running process instances to new process model versions. While the development organization can provide migration logic for their process models, it is not aware of models created by the customer. The reference process models provided by the development company are often a good starting point for customizations. This may lead to the desire to merge changes done by the development organization into derived local models.

Furthermore even unchanged business process models may break because of modifications to business functions: Consider the following simple process model consisting of two activities shown in Figure 2 a): "Apply for study" and "Decide application". The first activity consists of several business functions arranged in a wizard such as "provide school leaving certificate". An art academy may change the "Apply for study" activity to replace this business function with "upload work sample" as seen in Figure 2 b). In the mean time the development organization may modify a business function used in "Decide application" so that it requires the grade from the school leaving certificate (Figure 2 c). Figure 2 d) shows that a simple merge will result in inconsistent process instance because of the missing but mandatory grade although the process model itself was not modified.

Note that simple attempts to resolve the conflict are not adequate: Reintroducing the deleted business function does not comply with the desire business process of the art school. But deleting the step of calculating statistics violates the law.

Therefore an approach is needed that allows for automatic detection of issues and easy conflict resolution of business processes. It has to take into account changes to all layers from process models over activities down to business functions.

## 3.4 Data

Integration of data from the basic infrastructure (in this case HISinOne) and the specific software developed on top of this is also problematic. When looking at the HISinOne situation, we can differentiate three different kinds of situations.

- The university manages its own operational data such as students and lectures.

- HIS provides reference data like lists of postal codes, bank identification numbers and other data common across all customers.

- HIS provides default data, for example, types of hardship that may result in preferred admissions of appli-

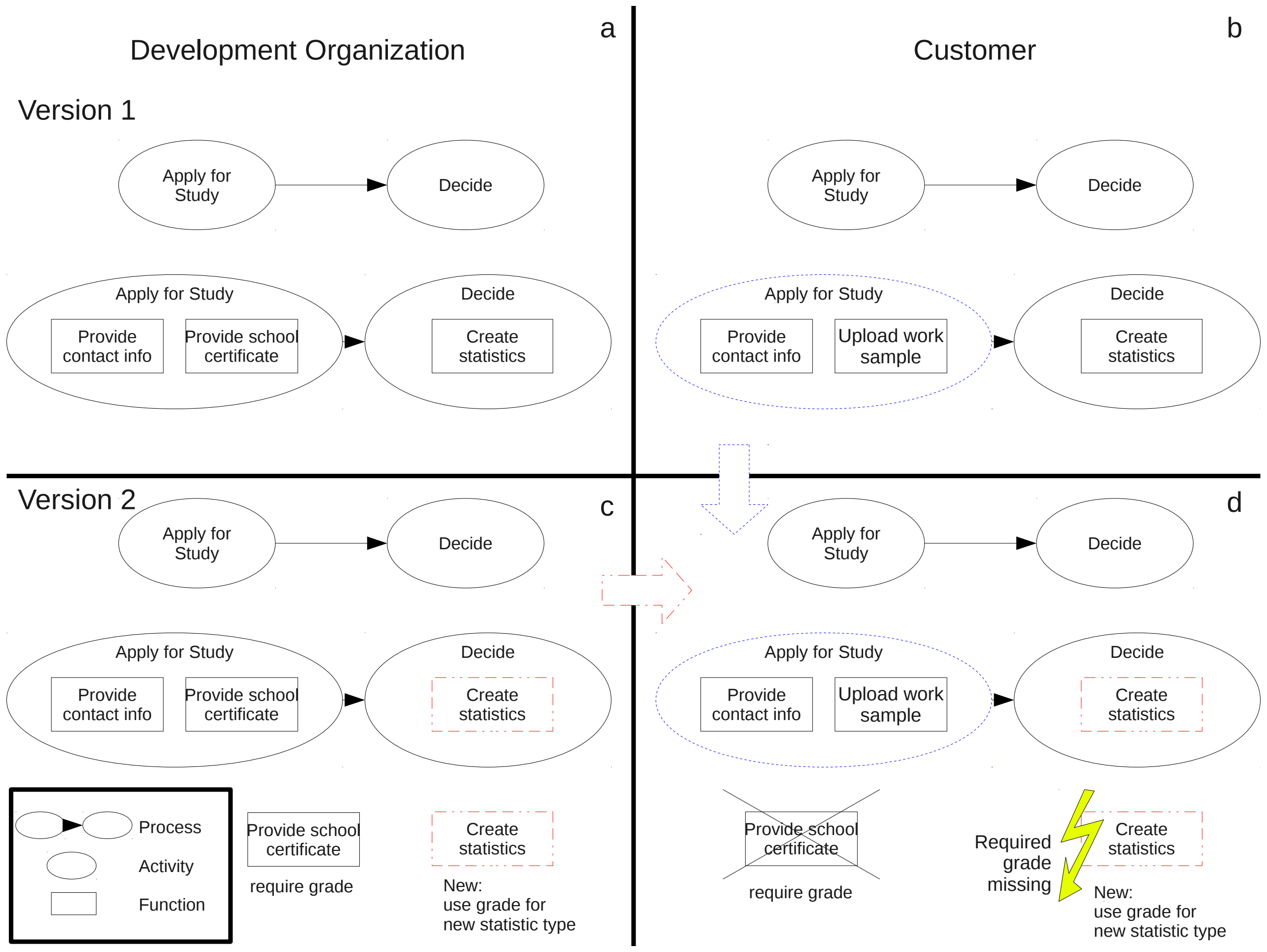

**Figure 2: Distributed evolution of activities and business functions a) Original version by the developing company. b) Modifications by the customer. c) New version by the development organisation. d) Merge Conflict.**

cants. Customers may add to, delete from or modify those lists.

All these forms of data can be regarded as variability as well because from a conceptual point of view there is no significant difference between a case where data controls a generic algorithm or where we have explicitly realized algorithm variations.

The first case in this list appears rather simple as the customer data itself need not be modified by any updates. However, even for this data changes to the data structure may occur. In HISinOne this may include adding attributes with default values or splitting attributes across classes. In case this happens update scripts take care of the data transformations. In cases where no change of the semantics happens, this is actually rather simple.

Updates to reference data is also a rather simple situation, as here no changes by the customer may occur. Thus, updates replace the existing data with new data that use the same object identifiers. Only in case of deletions special care needs to be taken in order to prevent loss of information in objects referring to the data in question. For example the country Czechoslovakia was split into Czech Republic and Slovakia. Therefore Czechoslovakia is not a valid option anymore and can be deleted from the list of countries. But it is desirable for people already stored in the system to keep the information instead of losing the origin completely. Therefore instead of actually deleting objects, they need to be flagged as invalid. HIS provides tools to search for objects that refer data marked as invalid.

The third case, however, is the most complicated. It requires that data is merged on updates which results in the following issues:

- The same attributes of the same object may have been changed.
- Different attributes of the same objects may have been updated by both the customer and the development organization in a way which results in an inconsistent state after merging.
- Objects may have been inserted by the customer that are provided as default data in a newer version of the product line infrastructure resulting in duplicated data.
- Objects may have been deleted by the customer and must not be added again on update.

A further issue comes from the way object references are typically handled in a relational database. It is common to use consecutive IDs to reference objects. For example, HIS may add a new right "may view own grades" which gets the next available id 42 assigned. Then it adds an entry in the role-rights table saying that the role "student" owns the right with id 42. In the mean time, however, a customer may have added a right called "may upload documents to elearning platform" which got the same id. This makes merging of updated variability information an extremely difficult process, as semantic integration of the variation needs to be ensured.

Merging of data requires a way to tell modified customer data and default data apart so that local modification are not reverted.

An approach is needed which allows for easy extension of data structures. It has to support all layers from the user interface to the data storage and allow validation against data types including enumerations.

## 4. CONCLUSION

In this problem statement, we focused on a specific mix of problems which has not yet received significant attention in the variability management community.

Some of the issues we looked at were driven by the fact that we took a closer look at information systems and differentiated the various forms of variability that may occur there. In particular, we looked at the individual difficulties in merging variations in user interfaces, data, business rules and processes.

Moreover, our analysis was driven by our specific case study context. The main complexity of variability handling is derived from the fact that updates are delivered to customers who may make their own modifications and additions in the mean time. This situation is typically referred to as open variability and sometimes as software ecosystems.

Here, we focused in particular on complexities of variability evolution that result from such a context.

This work is part of an ongoing effort to develop better technical and conceptual support for open evolutionary development of information system product lines.

## 5. REFERENCES

[1] J. Bosch. From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference (SPLC'09)*, pages 111–119, 2009.

[2] P. Clements and L. Northrop. *Software product lines: Practices and patterns*. Addison-Wesley, 2001.

[3] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21 ESD-90-TR-222, Software Engineering Institute Carnegie Mellon University, 1990.

[4] C. Krueger. Towards a taxonomy for software product lines. *Software Product-Family Engineering*, pages 323–331, 2004.

[5] K. Schmid. Variability modeling for distributed development — a comparison with established practice. In *Proceedings of the 14th International Conference on Software Product Line Engineering (SPLC'10)*, pages 155–165, 2010.

[6] K. Schmid and I. John. A customizable approach to full-life cycle variability management. *Science of Computer Programming*, 53(3):259–284, 2004.

[7] F. v. d. Linden, K. Schmid, and E. Rommes. *Software product lines in action: the best industrial practice in product line engineering*. Springer, 2007.

[8] R. van Ommering. Building product populations with software components. *Software Engineering, International Conference on*, 0:255, 2002.