Journal of Universal Computer Science, vol. 14, no. 8 (2008), 1207-1227
submitted: 15/11/07, accepted: 25/3/08, appeared: 28/4/08 © J.UCS

Embedded Software Revitalization through Component Mining and Software Product Line Techniques

Marcelo A. Ramos [1,2]
([1] ADC/LAC - Application Development Center for Latin America and Caribbean
VeriFone do Brasil
São Paulo, SP, Brasil
marcelo_r1@verifone.com)

Rosângela A. D. Penteado [2]
([2] DC - Department of Computer Science
Federal University of São Carlos
São Carlos, SP, Brazil
rosangel@dc.ufscar.br)

Abstract: The mining of generic software components from legacy systems can be used as an auxiliary technique to revitalize systems. This paper presents a software maintenance approach that uses such technique to revitalize one or more embedded legacy systems simultaneously and, in addition, create a core of reusable assets that can be used to support the development of new similar products. Software Product Line techniques are used to support the tasks of domain modelling and software component development. A real case study in the domain of Point of Sale (POS) terminals is presented and it illustrates the use of the proposed approach to revitalize three similar embedded legacy systems, simultaneously. It also shows how it is possible, through the created core of reusable assets, to deliver variations of these systems to meet the requirements of a wide family of POS terminals with different hardware configurations.

Keywords: Maintenance, Reuse, Component Mining, Gateway, Feature Model, Software Product Line, Hardware Decomposition, Embedded System.
Categories: D.2.13

1 Introduction

In the domain of embedded systems, hardware variabilities often happen in an unplanned way. Old components are often substituted by more current ones, which are cheaper and more efficient, thus producing improved versions of the original product. Possibly, some components were not even in existence when the first version of the product was delivered. Many companies of the embedded systems domain want the evolution of their software to occur with the partial or full reuse of the artifacts of their succeeded applications in order to create new ones with little or no maintenance [Graaf, Lormans and Toetenel 2003]. In this context, it is the responsibility of the software engineers to choose the most suitable development techniques and technologies to achieve this goal in a given domain for a particular set of new requirements. A brief example is given below.

A Point Of Sale (POS) device is a programmable electronic equipment with an embedded system and a reduced set of hardware peripherals, managed by an embedded operating system [VeriFone 2008]. It is mostly used to perform Electronic Funds Transfer (EFT) using payment cards. In this domain, hardware variability is a strong competitive factor since companies are expected to offer a wide range of solutions, from low cost to high-end technologies, depending on the needs of each customer [Visanet 2008]. Hardware variabilities are often designed as pluggable modules to enable fast product evolutions and to decrease deployment costs. Therefore, POS applications must be designed to follow hardware evolutions, and foresee emerging technologies.

Leveson and Weiss (2004) mention that many of the reports about success in software reuse practices have been premature because most of the artifacts created are abandoned due to their aging and degradation, even before a satisfactory return on investment is reached. In order to minimize these effects, they warn that the creation of reusable artifacts must be carried out carefully so as to enable easy and safe maintenance and also to expand the functionality of the systems they are built in.

The mining of generic components from legacy codes and their subsequent reconnection to the original systems is a technique that can support system revitalization by extending its functionalities. This can also allow the production of a core of reusable assets to support the development of new similar products. However, if legacy codes and components are developed through different paradigms, the development of gateways is necessary [O'Brien 2005], i.e., interface adapters that enable functions of the legacy code to easily access component interfaces.

This research offers a maintenance approach to revitalize one or more similar embedded legacy systems and to incrementally create a core of reusable assets, which can support the later development of other similar systems, members of a product family. Through the activities proposed, it will be possible to extend the life cycle of embedded legacy systems, enabling them to support new domain variabilities. The partial or total reuse of legacy artifacts can enable the quick creation of new systems, therefore, increasing the productivity and the competitiveness of the companies. In addition, this approach can lower the risk and cost of software development.

After this explanation, the paper is organized as follows: Section 2 contains a brief background, with a short description of the concepts, techniques and technologies included in the literature and which are relevant to the topic of this paper. Some of the related work is discussed in Section 3. Section 4 describes the proposed approach in terms of a framework of activities for mining reusable assets from legacy systems, and Section 5 describes the process used to perform the revitalization of such systems based on the artifacts created. A real case study, performed in the domain of Point Of Sale (POS) terminals, is illustrated in Section 6. Finally, the conclusions and the future work is discussed in Sections 7 and 8, respectively.

2 Background

Embedded systems are specialized computer systems made of hardware and software that integrate systems of wider functionality and that perform specific and predefined tasks. Software written for embedded systems is called embedded software or firmware. Despite the wide variety of projects, embedded software usually runs in hardware with a reduced set of resources and a limited capacity of processing and storing information [Vahid and Givargis 2002].

Software reuse refers to the construction of systems from existing artifacts, rather than developing them from scratch [Krueger 1992]. Different techniques have been researched with the purpose of increasing reuse to higher abstraction levels, for example: Components [Szyperski 2002] and Software Product Line [Clements and Northrop 2001].

Components are context-independent composition elements, implemented for a certain specification, distributed in an autonomous way and that, through their interfaces, add functionalities to the systems they integrate [Szyperski 2002]. However, in the domain of embedded systems, components are not usually autonomous elements (run-time components), but codes written in high-level languages, that can be connected to the code of an application during the creation of system versions (build-time components) [Crnkovic 2005]. Easily-adaptable generic components can support the creation of reusable solutions for similar requirements in different domains. For such, they have to support the commonalities and variabilities of the domains to which they are applied [Bergey, O'Brien and Smith 2000].

Software Product Line (SPL) consists of a group or family of products that share a common architecture and belong to a particular domain. The purpose of SPLs is to increase the efficiency of development processes by exploring the identification and reuse of commonalities and managing variabilities of related products [Atkinson et al. 2002]. Figure 1 shows a generic SPL engineering process comprised of two main activities: Domain Engineering creates the core of reusable assets and the SPL development infrastructure, and Application Engineering develops new products, family members, from the available resources [Ziadi, Jézéquel and Fondement 2003]. The core of reusable assets of a SPL contains its requirements, domain models, architecture, software components etc.

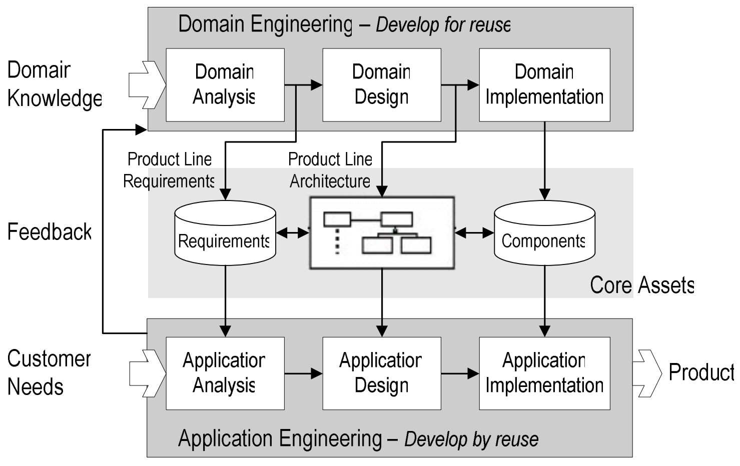


Figure 1: Generic SPL engineering process [Ziadi, Jézéquel and Fondement 2003]

The process of mining assets often refers to the non trivial activities of uncovering and extracting potentially useful and reusable artifacts built into the legacy systems of a company [Bergey, O'Brien and Smith 2000]. Automation tools can be designed for this purpose, such as the one developed by Eisenbarth and Simon (2001). In general, handling such tools requires the work of experienced software engineers to properly analyze the results and refine mining rules.

Features are properties of a domain, visible to the user, that enable the identification of commonalities between related systems as well as their variabilities. With the purpose of improving the identification of important or special properties of a domain during the analysis phase, Kang et al. (1990) introduced the feature model in their FODA method (Feature Oriented Domain Analysis). In this model, the features are arranged hierarchically in a tree structure where they are connected by structural relationships, forming groupings. Each feature has its own specifier that defines it as mandatory, optional or alternative. Different notations have been proposed to extend the representative aspect of the feature model in relation to different types of structural relationships. For example, Czarnecki and Eisenecker (2000) differentiate XOR relationships, between mutually exclusive alternative features, from OR, between optional features. Riebisch et al. (2002) expand the OR relationships with UML multiplicities [Booch, Rumbaugh and Jacobson 1999] increasing the range of possible combinations among optional features. Figure 2 illustrates an extended feature model, in which a reader must inform his/her personal information, such as his/her name and two or more optional pieces of information such as address, birthday, ID and mother's name.

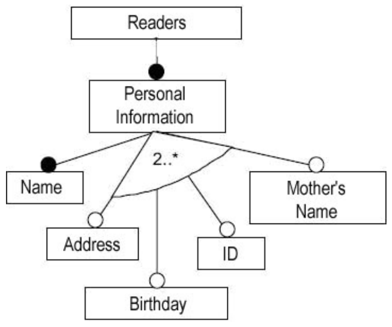


Figure 2: Extended feature model

Gomaa (2004) uses an alternative form to model the features of a domain, based on the UML. In this way, it is possible to make use of the available UML resources and modeling tools. However, not all of the graphic notations have a corresponding element in UML and they are commonly represented through custom stereotypes, e.g., <<kernel>>, <<optional>>, <<variant>> etc. Figure 3 shows Gomaa’s conceptual static model for a microwave oven software product line.

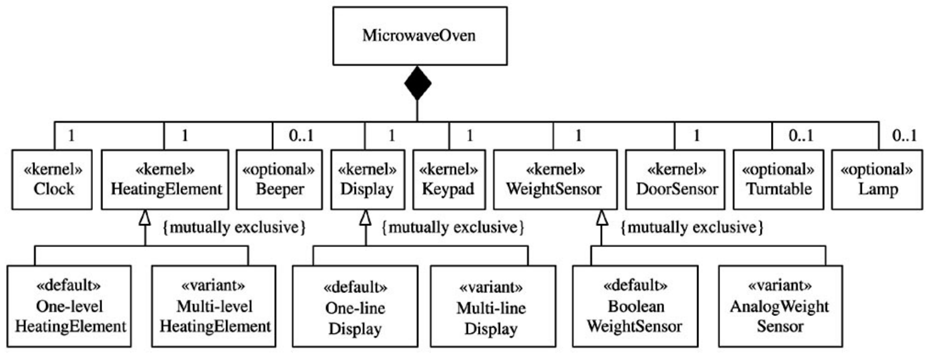


Figure 3: Conceptual static model for a microwave oven software product line [Gomaa 2004]

3 Related Work

Many known methods for the creation of SPL propose the advanced planning of all products that can be developed with basis on a common development infrastructure to be created for a particular family. They do not, however, mention clearly how to support unplanned variabilities, like the hardware variabilities mentioned in Section 1. Furthermore, many of these methods do not mention the possibility and the way in which the SPL is created with basis on the legacy artifacts of a domain. For this reason, they were not included in this section.

Methods for the creation of SPL in embedded system domains usually include an initial reengineering activity to create reusable assets from legacy systems, for the future development of a family of products. An example is the FOOM method (**F**eature-based **O**bject **O**riented **M**odeling) [Ajila and Tierney 2002]. However, potentially complex tasks, such as architecture recovery and transformation, are still fully performed before a single variability, even a simple one, becomes available as a new product. This fact is also found in other methods for SPL asset recovery from legacy systems, like the one proposed by Kang et al. (2005). An iterative method that enables an incremental addition to the legacy code of support to variabilities of the domain, according to the particular and immediate needs of each company, is an alternative approach to reduce time, risks and costs associated to the creation of product families. Some research that uses this approach are described briefly below.

Eisenbarth and Simon (2001) use computational tools to support the incremental recovery of legacy architectures and code validation. These tools also support the recovery of features from legacy codes by creating a map of their internal connections to enable the design of reusable asset interfaces. According to them, fast and cheap reuse can lead companies to adopt SPL for the development of their new products.

Mehta and Heineman (2002) propose a methodology for the modernization of software systems that combines features, regression tests and component-based software engineering. Their proposal foresees the modernization of a group of features identified during regression tests. The code associated to each feature is identified, refactored, converted into a software component and inserted again into the application. These components can be reused later in other applications.

Bergey, O'Brien and Smith (2000) propose that the mining of assets for SPL be performed in four steps: 1) gather preliminary information, 2) analyze the feasibility of mining the assets and define its strategy, 3) obtain detailed technical understanding of the existing assets and 4) recover the assets. Bosch and Ran (2000) consider essential the creation of a feature model to support these tasks.

This paper uses the mentioned research as a basis for achieving embedded software revitalization. Feature modeling is a proven method to support domain analysis and can facilitate the elicitation of hardware commonalities and variabilities. Likewise, hierarchically distributed features are suitable elements to support incremental mining of assets for SPL. Another important fact to consider is that many companies in the domain of embedded systems are hardware manufacturers rather than software developers. Usually, they have a small software development staff which is often not very familiar with software development frameworks that contain various management activities and hierarchical levels. Therefore, the mining of assets using the few steps proposed by Bergey, O'Brien and Smith (2000) should be enough for most of the real embedded system environments.

4 A Framework of Activities to Mine Reusable Assets from Legacy Systems

Usually technologies for traditional software development do not consider the specific needs associated to the creation of embedded software and the usual constraints of this domain, such as memory limitation, power consumption and hardware changes [Graaf, Lormans and Toetenel 2003].

The framework of activities proposed herein foresees the creation of software components from features built into embedded legacy systems. For such, the feature mining process is based on an adaptation of the four steps proposed by Bergey, O'Brien and Smith (2000), described in Section 3, and supported by feature models, as proposed by Bosch and Ran (2000).

The adapted process model is presented next. It considers the availability of legacy source codes and also the minimal documentation of the existing hardware and software elements, i.e., peripherals, operating system etc.

Step (1): The process begins with a meeting that brings together programmers, users and other people directly or indirectly involved with the legacy systems. The purpose of the meeting is to identify the current deficiencies and the immediate and future requirements of the systems.

Step (2): Through the information obtained and customer support, it is possible to come up with a preliminary analyses of the technical and economical feasibility of the project to revitalize the systems, based on the previously identified requirements. Once the decision has been made to give continuity to the project, these requirements serve as a guide for outlining a suitable strategy for the development of the project, as well as for the establishment of its scope, goals and priorities.

Step (3): This step starts with the creation of a group of technical people, that include one or more domain specialists, to better understand the legacy systems and their documentation and to analyze, as best as possible, the specific concepts of the domain and its particularities. The priorities of the project help to define the set of knowledge to be acquired initially, which can involve the entire system or just part of it, depending on whether the mining is to be carried out fully or gradually. SPL techniques for domain modeling are used to document the knowledge obtained in this stage and to support the next mining step of the process.

Step (4): Based on the information acquired and documented up to this point, the mining activities begin and may vary depending on the goal of the project and on the previously established strategy.

A feature model, in which each feature is associated to a single component, is used to model the domain and to provide support for the future development of generic software components. The divisions and junctions of features, Figure 4, described by Sochos, Riebisch and Philippow (2006) in their method FArM (**F**eature-**Ar**chitecture **M**apping), are removed through successive refinements made to the model.

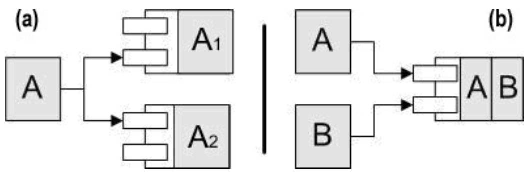


Figure 4: (a) Division of the feature A in two components A1 e A2;
(b) Junction of the features A and B in a single component AB;
[Sochos, Riebisch and Philippow 2006]

In a gradual component mining process, the feature model of the domain can be simple at first, just detailing the features that represent immediate requirements and current deficiencies of the legacy systems. As the components are developed, refinements and additions of new features can be made to the existing feature model. This process is performed iteratively.

The legacy code serves as a guide for the design of the generic interfaces of the mined components, which must implement, at least, the actual system functionalities. Techniques such as inheritance and configurable interfaces can be used to extend the system functionalities through the support of domain variabilities. For every feature of the feature model, an inspection in the legacy code must be carried out to identify all the functions directly related to it. Based on the properties of these functions, a Connection Map (CMap) is built and used to back the design of the generic interfaces of the component related to the feature. When there is a group of similar systems, CMap can map the connection of their codes with the common feature in a unified manner. This increases the generality of the component interface in development, widening the possibilities of its reuse in new products. Figure 5 shows, in a simplified model, the activities of the process described above and its four steps.

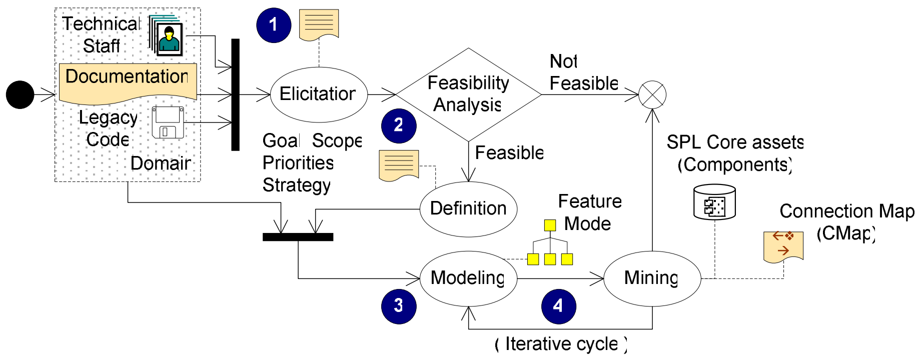


Figure 5: Process model to mine reusable assets from legacy codes

The format of the Connection Map (CMap) is presented in Table 1 and the columns refer to its elements.

Feature:		**<Feature ID>**			
Function	**Group ID**	**Parameters**	**Return**	**Comments**	**Applications**
[Constraints] **[New Requirements]** **[Observations]**					

Table 1: Connection Map (CMap) and its elements

Feature ID: Feature name in the feature model. In order to facilitate its location, an optional extended ID can be supplied, which include all the features that precede the current one in the tree branch it belongs to. For example, the feature B child of feature A, can be identified by A→B;

Function: Function of the legacy code that fully or partially implements the related feature;

Group ID: Unique number to identify distinct groups of functions. If similar functionalities are performed by different functions in different applications, these functions must be assigned to the same group ID when inserted into the CMap. The revitalized code must provide a generic component interface that supports all variations documented in the group;

Parameters: List of function parameters. The objective of each parameter should be easily identified through its description, written in the form < name > | <data type > as in the UML. For functions without parameters, “None” is used.

Return: Function return. It follows the same rule described above for Parameters;

Comments: It contains a brief description of the functionality implemented. It can also contain relevant information to help in the feature mining of the legacy codes;

Applications: Each similar legacy application of the domain receives a unique ID at first. Once a function is identified within one or more of these applications, the IDs of these applications will appear in this column of the table. This fact frequently occurs when the adopted reuse technique involves code sharing or duplication. This list indicates if feature mining involves multiple applications;

Constraints: During feature mining, technical or functional limitations of any kind, may appear and should be documented in this section and taken to the domain specialists, who will decide on the best solution for them. Specific meetings with experts can be scheduled to analyze and solve potential issues;

New Requirements: When the solution for a particular feature limitation has to be addressed by the component, its description is appended in this section and it will become a part of the requirements for the component implementation.

Observations: Free text area to facilitate the communication of the team members who are mining a common particular feature. This can include the current status of the task, a list of pending issues etc.

The CMap creation enables a wider understanding of the legacy code through the accomplishment of code inspections, which evaluate the coupling level among the code and each of the features to be mined, facilitating the performance of impact analyses. Specific methods, such as the ones proposed by Mehta and Heineman (2002) and by Eisenbarth and Simon (2001), can be used to identify functions of the legacy code that implement a particular feature. In this way, the process model, proposed to mine reusable assets from legacy codes, Figure 5, can be used as the framework for generic activities that can be adapted according to the properties of both the domain and the operational environment.

The following section describes an approach to revitalize legacy codes based on the process described herein.

5 Legacy Code Revitalization through the Mining of Assets

The resulting artifacts of the asset mining process, described in the previous section, are: a) a feature model, that models the domain features, b) a CMap, that documents the connection of these features with the legacy systems belonging to the domain and c) a core of reusable software components, whose generic interfaces implement, at least, the current functionality of the legacy systems. Although the components can be used independently to create new products, they also can, in the proper manner, be reconnected to the legacy code to improve its structural organization and to aggregate new functionalities to the application, extending the life of the original product. This revitalization approach considers the implementation of gateways, as proposed by O'Brien (2005), which enable the use of different paradigms for designing components.

The CMap and the component interface documentation, produced by the asset mining process, supply enough information to implement gateways that act as interface adapters. In order to rebuild the original systems using the recently created artifacts, legacy functions, now implemented as component methods, must be removed from the legacy code. However, the original function calls, which will be redirected to the components by the gateways, must be maintained.

In this manner, legacy systems revitalization is obtained without making any changes to the code structure and without interfering in the daily activities of the maintainers of the systems. Figure 6 shows the process described, highlighting how the mined components can extend original system functionalities or enable the creation of new similar products.

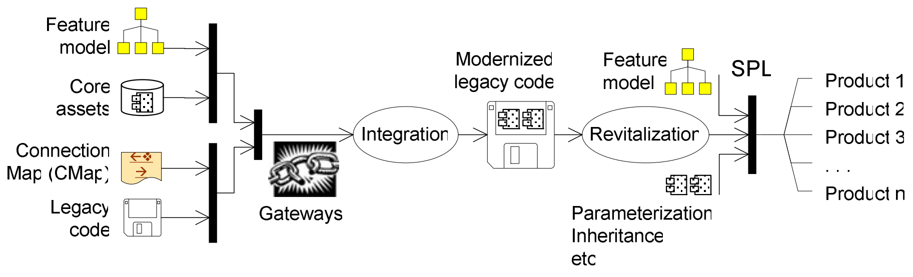


Figure 6: Legacy code revitalization through the mining of assets

The revitalization, when performed gradually, can facilitate the execution of validation tests of the rebuilt systems, since the changes are focused on isolated features and are entirely implemented in the components. The same existing test cases for the legacy system can be used for the revitalized one, to check if the functionality has actually been maintained. A case study is shown below to exemplify the revitalization process described hereby.

6 Case Study

Point Of Sale (POS) terminals are small, portable and versatile devices with a reduced set of standard peripherals, managed by an embedded system with a microprocessor and an operating system (OS). The POS terminals are designed to meet the needs of a wide market segment. Nevertheless, due to their security capabilities, they are mostly used for Electronic Funds Transfer (EFT) with payment cards. Figure 7 shows a POS terminal and its main hardware peripherals.

In the domain of POS terminals, hardware variabilities enable the simultaneous offering of a wide range of similar products to the market, which is an important business strategy for companies. However, strict certification processes of EFT applications require that the business rules, built into the legacy codes, remain unchanged and free of errors during the creation of different versions of the applications. In this context, it is common to fully reuse the legacy applications, created through old development paradigms, in order to create new ones, even with the documentation and source code available degraded by successive maintenances. In general, domain variabilities are implemented through the conditional compilation of blocks of code, which makes the reading and understanding of the code progressively more difficult. Domain specialists are common in these environments and help with many of the activities proposed in this work.

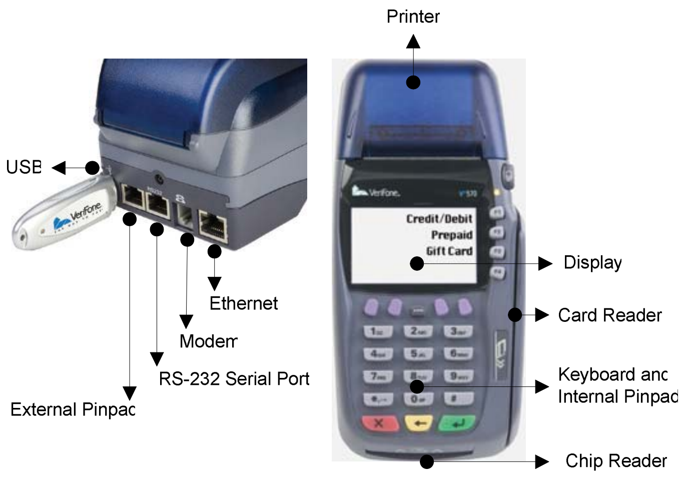


Figure 7: POS terminal and its main hardware peripherals[1].

For this case study, three similar legacy EFT applications, developed for POS terminals and belonging to a real company, were used. These applications are different, have modular structure, source codes written in C with unsatisfactory reuse levels and minimal documentation. Existing variabilities are related to frequent technological evolutions of the hardware and were implemented as conditional blocks of code, named Selectable Modules[2]. The goal of the company is to quickly rebuild each of the legacy applications for different, yet similar, hardware platforms, while maintaining their business rules, which have already been certified by strict certification processes and should not be modified. Experienced programmers of the company acted as domain specialists in the process. The asset mining process and the legacy codes revitalization from the created artifacts were performed as follows.

A startup meeting (Step 1), involving strategic areas of the company, was held to find the deficiencies and the immediate and future requirements of the three applications, according to the perception of each participant. After the facts and artifacts were gathered, a preliminary analysis of the technical feasibility of the revitalization project (Step 2) was carried out.

Due to the requirement to maintain the business rules and support hardware variations, it was decided to gradually and concomitantly revitalize the applications by focusing on features of the hardware, beginning with variabilities of low operational impact to facilitate the preliminary validation process with basis on the first results.

Once the strategy (Step 3) was defined, the study of the domain began with the participation of programmers appointed by the company. After reading the documentation, different legacy codes were analyzed and doubts were cleared up to

[1] VeriFone Vx-570 POS terminal [http://www.verifone.com.br]

[2] Internal denomination only, created by the programmers of the company.

better understand the hardware elements and their usage by the applications. The know-how obtained was documented as a feature model, shown in Figure 8.

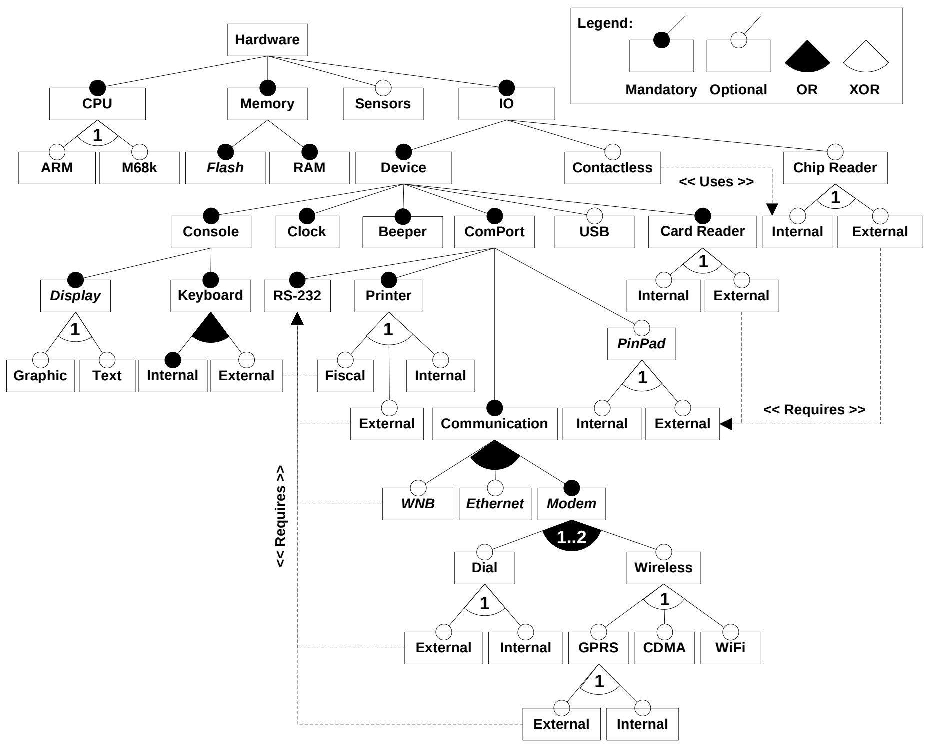


Figure 8: Hardware feature model of POS terminals.

Physical and logical features of the hardware were combined in the feature model to represent not only the physical constitution of the POS terminals, but also their logical structure from the point of view of the operating system. One of the results obtained, for example, was the console (logical feature) comprised of display and keyboard (physical features). This combination was also intended to make mapping the feature model to UML class and component models easier.

The activities to mine reusable assets, described in Section 4, were started (Step 4) based on the strategy defined previously. Since the legacy applications do not support an optional external keyboard and since the keyboard is a hardware variability of low operational impact and important for the company's business strategy, it was decided to validate the process starting at its implementation. Inspections were performed in the legacy codes of the three available applications to identify the lowest level functions linked directly to the chosen feature. Low-level functions generally handle the hardware directly and do not include business rules, so they were chosen as the starting point for the asset mining process. In order to support the development of the associated component, information about the desired behavior of the feature and important limitations of the internal and external keyboards were gathered and registered as an extension of the CMap, shown in Table 2.

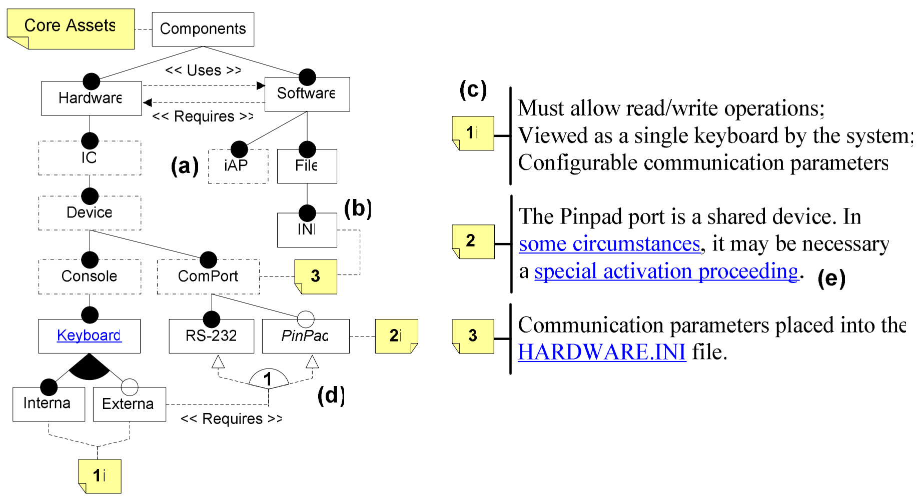
Feature:		**IO → Device → Console → Keyboard**			
Function	**Grupo ID**	**Parameters**	**Return**	**Comments**	**Applications**
KBHIT	#0	None	status: bool	TRUE if key pressed.	#1, #2, #3
get_char	#1	None	key : int	Wait for key pressed and return it.	#1, #2, #3
[Constraints] . OS does not allow writing operations to the internal keyboard buffer. . OS does not provide an external keyboard buffer. **[New Requirements]** . Internal and External keyboards must share a common buffer provided by the component, which must allow read/write operations. . External keyboard can be optionally connected to the POS terminal at the ports RS-232 or External PinPad, in a mutually exclusive way. The communication parameters of such ports must be configurable.					

Table 2: Keyboard feature Connection Map (CMap)

Component development started with the instantiation of a feature sub-model derived from the hardware feature model to separate just the convenient features. Figure 9 shows the created feature sub-model, which contains refinements made to the original model to meet the new requirements described in the CMap. They are: a) a dependence of the external keyboard in relation to the feature PinPad and, b) two new software features, not existing in the legacy code, the iAPI (Figure 9a), to increase the portability of the created artifacts and the INI type files (Figure 9b), to store the communication parameters.

In the initial feature model, Figure 8, domain specialists noticed that important information, linked to some features of the hardware, could not be clearly represented using the available modeling elements. To bypass this deficiency, a simple feature model extension was proposed. It was named Note (Figure 9c) and its description is given below.

Notes: They are represented just like in the UML and can be connected to one or more features through anchors. For legibility reasons, they have an internal sequential number that refers to the explanatory texts, external to the feature tree. In general, Notes enrich the feature model with domain details that cannot be represented graphically but play an important role in decision making. In high abstraction levels of the feature model, notes can also point to temporary information, which can guide refinement activities in later stages of the project. When the text referenced by a Note contains implementation advice, its internal sequential number is followed by the letter 'i'.



Must allow read/write operations;
Viewed as a single keyboard by the system;
Configurable communication parameters

The Pinpad port is a shared device. In some circumstances, it may be necessary a special activation proceeding.

Communication parameters placed into the HARDWARE.INI file.

Figure 9: Feature Sub-Model for the keyboard component

Additionally, specifiers and multiplicities were also applied to dependency relationships between features, identified by stereotypes as <<Requires>>, <<Uses>> etc., similar to UML (Figure 9d). In this way, it is possible to model the dependency of the external keyboard in relation to the communication ports RS-232 and Pinpad, emphasizing their alternative use. To represent the logical elements of the hardware, differentiating them from the visible features, an extension named Sub-Feature was also proposed.

Sub-Features: These are represented by dotted outlines (Figure 9a), indicating that they are not visible to the user but are important as structural elements to support a more realistic modeling of the domain. Sub-Features identification activity was named feature pulverization, since it includes one or more basic foundation elements.

Hyperlinks (Figure 9e) were attached to some elements of the feature model to establish a link between them and other available external documents, which may contain relevant information.

In the feature sub-model of Figure 9, Internal and External Pinpad features were not represented, since just the serial communication capability of the PinPad feature, provided by the ComPort feature, must be implemented.

The mapping to a UML class model was performed with basis on the information contained in the CMap and in the operating system documentation, which describes the services available to make use of most of the hardware devices. In the class model of Figure 10, the internal and external keyboards were unified by the `Keyboard` class that allows read/write operations to its circular queue, according to the project requirements.

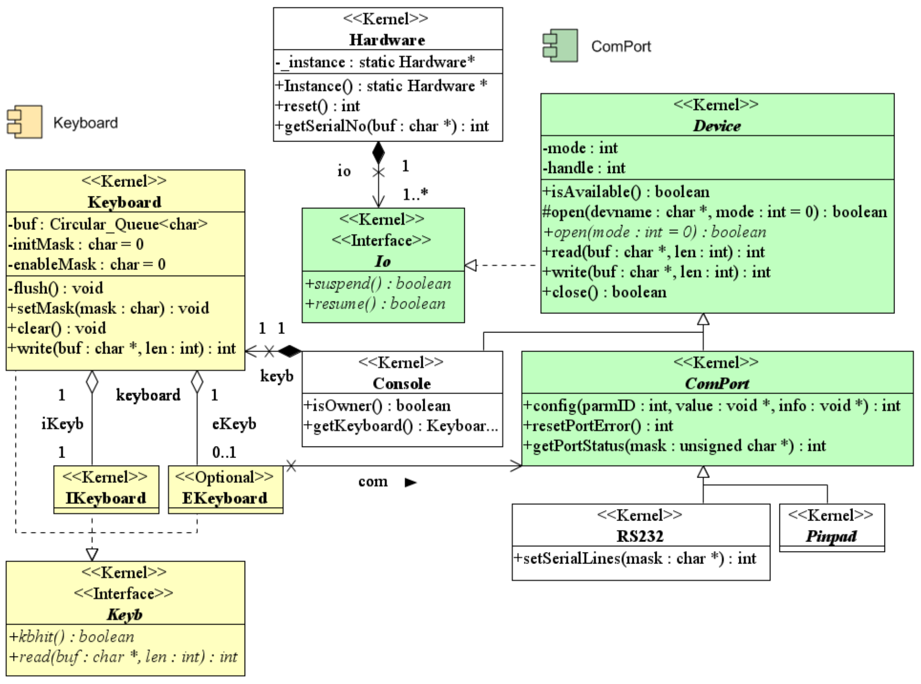

Keyboard
ComPort
<<Kernel>> Hardware
-_instance : static Hardware*
+Instance() : static Hardware *
+reset() : int
+getSerialNo(buf : char *) : int
<<Kernel>> Keyboard
-buf : Circular_Queue<char>
-initMask : char = 0
-enableMask : char = 0
-flush() : void
+setMask(mask : char) : void
+clear() : void
+write(buf : char *, len : int) : int
<<Kernel>> <<Interface>> Io
+suspend() : boolean
+resume() : boolean
<<Kernel>> Device
-mode : int
-handle : int
+isAvailable() : boolean
#open(devname : char *, mode : int = 0) : boolean
+open(mode : int = 0) : boolean
+read(buf : char *, len : int) : int
+write(buf : char *, len : int) : int
+close() : boolean
<<Kernel>> Console
+isOwner() : boolean
+getKeyboard() : Keyboar...
<<Kernel>> ComPort
+config(parmID : int, value : void *, info : void *) : int
+resetPortError() : int
+getPortStatus(mask : unsigned char *) : int
<<Kernel>> IKeyboard
<<Optional>> EKeyboard
<<Kernel>> RS232
+setSerialLines(mask : char *) : int
<<Kernel>> Pinpad
<<Kernel>> <<Interface>> Keyb
+kbhit() : boolean
+read(buf : char *, len : int) : int
io
1
1..*
keyb
keyboard
iKeyb
eKeyb
0..1
com

Figure 10: Keyboard feature class model

A method to set up communication parameters, in accordance with the same group of requirements, was added to the ComPort class. The class for handling INI type files, not included in this paper, implements the reading of sessions, delimited by brackets, and stores their configuration keys in the memory and enables the query of their values. A component model was developed from the class model and it documents the services provided by the created core of reusable components. Figure 11 shows, in a simplified model, the first reusable components developed.

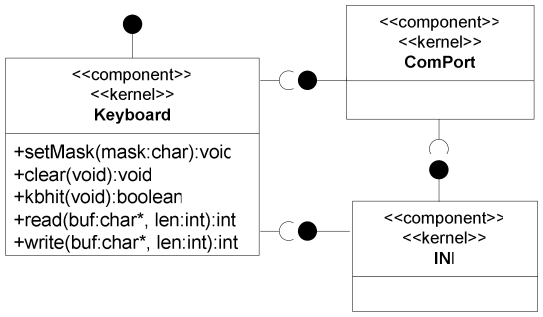
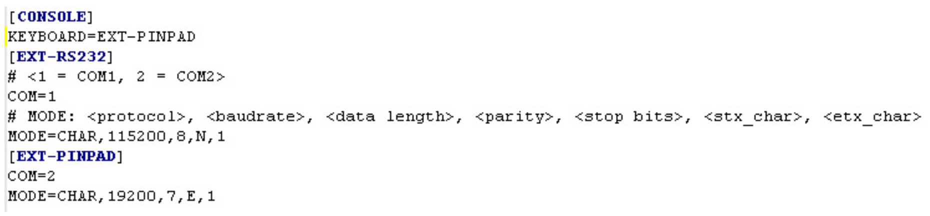

<<component>> <<kernel>> Keyboard
+setMask(mask:char):voic
+clear(void):void
+kbhit(void):boolean
+read(buf:char*, len:int):int
+write(buf:char*, len:int):int
<<component>> <<kernel>> ComPort
<<component>> <<kernel>> INI

Figure 11: Keyboard feature simplified component model

Figure 12 shows the `hardware.ini` configuration file, with two connections available for the external keyboard, EXT-RS232 and EXT-PINPAD, in accordance with the feature requirements. In this file, the external keyboard was linked to the EXT-PINPAD port and has to operate according to the parameters defined by the MODE key of this session. If an empty value is assigned to the KEYBOARD key, the external keyboard is disabled.



```
[CONSOLE]
KEYBOARD=EXT-PINPAD
[EXT-RS232]
# <1 = COM1, 2 = COM2>
COM=1
# MODE: <protocol>, <baudrate>, <data length>, <parity>, <stop bits>, <stx_char>, <etx_char>
MODE=CHAR,115200,8,N,1
[EXT-PINPAD]
COM=2
MODE=CHAR,19200,7,E,1
```

Figure 12: Hardware.ini configuration file.

Once the components were created, the gateway implementation was started, guided by the content of the CMap. Since the listed functions are direct calls to the OS API (Application Programming Interface), the respective implementations of the legacy code did not have to be removed. However, due to the conflict of names, they were renamed and received a prefix _G_ to identify that they were implemented in the gateway. No other change was made in the legacy code of the applications.

The developed gateway for the feature `Keyboard` is shown in Figure 13. It also shows how a useful resource of the text editor was used to query the references of an internal function of the legacy code. Keyboard access occurs through the Console, following the hierarchy modeled in the feature sub-model.

Since there is no explicit initialization of the `Keyboard` component by the application, it occurs through the gateway with a call to the function `initConsole()` when the first call of any method of the component interface occurs. At this moment, an instance of the keyboard component is created, enabling access to the internal keyboard and, optionally, to the external keyboard, depending on the configuration contained in the `hardware.ini` file.

Figure 14 shows an example of how to replace functions in the legacy code for corresponding ones, implemented in the gateway. This example also shows that, although the component keyboard has a `clear()` method that clears the content of the keyboard buffer, the function `vdKeyFlush()`, that has the same purpose, was maintained, given that the function `fDeactivated()` implements a business rule that must be maintained, according to the project requirements.

Once the gateway was built, it was compiled with the legacy codes of the applications and with the implementation of the components. As a result, the legacy applications could be run with both internal and external keyboards and the business rules were maintained, according to the initial requirements.

```
using namespace std;

#include "Console.h"
#include "Keyboard.h"
#include "Sdk.h"
#include "Gateway.h"
namespace Hw
{    class Console   ;
     class Keyboard  ;
}
Hw::Console*     _c   = NULL  ;
Hw::Keyboard*    _k   = NULL  ;

void initConsole(void)
{    /* Defines static variability control through hardware.ini configuration file */
         _c = new Hw::Console( "HARDWARE.INI", "CONSOLE" );
     /* Address the Keyboard component .. */
     /* .. configured by the key "Keyboard =" in the [CONSOLE] section */
     _k = _c->getKeyboard();
}
int _G_kbhit  (void)  /* Keyboard MCo's KBHIT() function gateway */
{        /* Console owns the Keyboard and must be initialized firstly */
         if  (   !_k )initConsole( ) ;
     return  (int)_k->kbhit( )        ; /* Redirect the call to the Keyboard component */
}
int _G_getchar(void) /* Keyboard MCo's get_char() function gateway */
{
     char     ch                      ;
     while(!_G_kbhit())               ; /* Once a key is detected ..   */
                _k->read(&ch, 1)      ; /* .. reads it                 */
     return  (int)    ch              ;
}
```

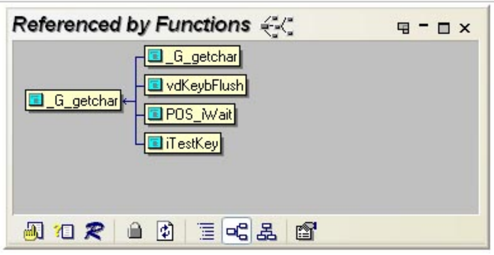
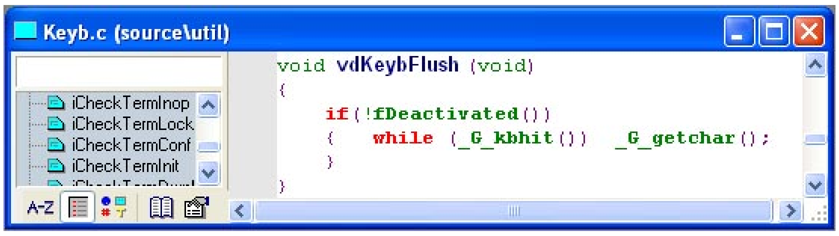


Figure 13: Gateway of the Keyboard feature



```
void vdKeybFlush (void)
{
    if(!fDeactivated())
    {    while (_G_kbhit())   _G_getchar();
    }
}
```

Figure 14: Example of replacing functions in the legacy code by the corresponding ones, implemented in the gateway

With a small design change it was possible to add an optional barcode reader to the system as an extension of the keyboard feature in order to speed up some of the existing payment transactions. Figure 15 shows the product family available after the first stage of the revitalization process.

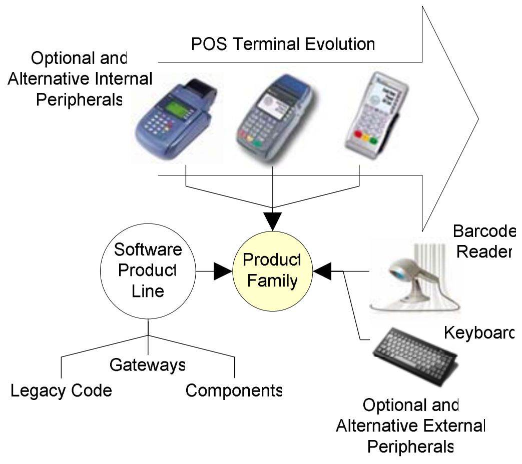


Figure 15: Product family available after the first stage of the revitalization process

The results were presented to the customer, who approved the continuity of the revitalization process for the creation of reusable components for all other hardware features. Extensions of the communication and security capabilities are expected by the customer.

7 Conclusions

The concluded case study shows how hardware variabilities can be managed through parameterized software components, which are isolated from the systems' business rules and connected to their legacy codes through gateways. It also exemplifies the use of mining generic components from legacy codes to revitalize multiple similar applications, concomitantly. Even after the performance of an incomplete but preplanned revitalization, the reuse level was improved significantly.

Although we believe that the approach presented can be applied to conventional personal computer systems, embedded software has facilitative factors. Generally, they are smaller and tasks like architecture recovery and code inspections tend to be less complex. Additionally, component technologies for embedded systems tend to be simpler since constraints like memory limitation may restrict the use of more modern ones, like Javabeans[3].

[3] Portable, platform-independent component model written in the Java programming language

For the domain in which the case study was applied, the performed process presented the following enhancements in relation to the more known proposals for the creation of SPL:

- Domain and application engineering phases are iterative, delivering the first results faster;
- Legacy codes are fully reused;
- Documentation is recovered gradually;
- Reusable assets are created incrementally with lower risk and cost;
- Mined components, once connected to the legacy code, can be tested in a real environment right after their implementation.

These enhancements, however, may be associated to specific properties of the domain such as the hardware variability and the existence of legacy source code and domain specialists.

The Unix-like structure of the OS of the used POS terminals does not allow the applications to directly access the hardware resources and they have to go through an Application Programming Interface (API). This particularity facilitated the construction of the CMap, reducing the need for interpretations of the legacy code.

The unified support to the internal and external keyboards using the `Keyboard` component revealed a situation in which the junction of features, Figure 4(b) was useful to meet the feature requirements, which contradicts one of the guidelines of the FArM method [Sochos, Riebisch and Philippow 2006].

Although the technique proposed by Mehta and Heineman (2002) was applied to a different domain, it can be adapted for POS terminal domains, which contain the particular previously-described features. The following improvements were made to the technique mentioned above: 1) besides modernizing the legacy code, through the extraction of features and the insertion of equivalent components, it was possible to revitalize it, extending its functionality to meet the new requirements; 2) the set of created components enabled not only the reuse of the features of the legacy system in other applications, but also the implementation of some domain variabilities in the legacy code, promoting the fast creation of a product family; 3) the use of feature models, instead of regression tests to support the mining of features of the legacy code, allowed a wider visualization of commonalities among different applications and of variabilities of the domain, enabling the creation of more generic component interfaces and the concomitant revitalization of similar products; 4) the gateways used to isolate the components from the legacy code allowed the use of more modern development paradigms to create those components and it also made the revitalization process imperceptible to the system maintainers, who could continue performing their usual maintenances; 5) the gradual revitalization of the legacy codes was performed with little risk to the customer, who was able to evaluate the costs and efficiency of the process in advance, through the first partial results delivered.

In relation to the feature mining process supported by computational tools, such as the one proposed by Eisenbarth and Simon (2001), the activity of code inspection had the following advantage: the mapping of interconnections between a feature and the legacy code, for the design of the corresponding component interface, is done according to particular selection rules established by the requirements of the project. In the case study shown, for example, the keyboard handling functions that implement any of the business rules were not considered in the development of the `keyboard` component, e.g., `vdKeybFlush()`, Figure 14. Such selection rule is difficult to be applied using automated processes and may require additional effort and expertise for the refinement of the partial results obtained.

8 Future Work

During the mapping of the feature model to classes of objects and components the occurrence of certain repetitive solutions was noticed, for which future research will be made to identify possible patterns that can facilitate the creation of tools to support feature-to-component mapping processes. For the purpose of confirming the generality of the proposed approach, it should be validated for the revitalization of legacy applications in different domains.

Generative and configuration management tools are being built to facilitate the creation and maintenance of product families. Alternative techniques for mining features from legacy codes will also be researched for performing equivalence tests for product families. The tools used by Eisenbarth and Simon (2001) will be applied to the case study described in Section 6 to verify how efficient they are in the extraction of a component similar to the one obtained in this work from the same legacy code used.

References

[Ajila and Tierney 2002] Ajila S.A.; Tierney P.J.: The FOOM Method - Modeling Software Product Line in Industrial Settings. Proceedings of the International Conference on Software Engineering Research and Practice, 2002, Las Vegas, USA.
<http://www.sce.carleton.ca/ faculty/ajila/SERP02 - Camera Ready.pdf>.

[Atkinson et al. 2002] Atkinson, C.; Bayer, J.; Bunse, C.; Kamsties, E.; Laitenberger, O.; Laqua, R.; Muthig, D.; Paech, B.; Wust, J.; Zettel, J.: Component-Based Product Line Engineering with UML. 1st Edition, 2001, Addison-Wesley Professional, USA.

[Bergey, O'Brien and Smith 2000] Bergey J.; O'Brien L.; Smith D.: Mining Existing Assets for Software Product Lines. Technical Note, CMU/SEI-2000-TN-008, 2000, SEI, USA.

[Booch, Rumbaugh and Jacobson 1999] Booch, G.; Rumbaugh, J.; Jacobson, I.: The Unified Modeling Language Reference Manual, 1999, Addison-Wesley, USA.

[Bosch and Ran 2000] Bosch, J.; Ran, A.: Evolution of Software Product Families. Proceedings of the International Workshop on Software Architectures for Product Families. 2000, LNCS, v. 1951/2000, p. 168-183, Springer-Verlag, UK.

[Clements and Northrop 2001] Clements, P.; Northrop, L.: Software Product Lines: Practices and Patterns. 1st Edition, 2001, Addison-Wesley Professional, USA

[Crnkovic I. 2005]. Component-based software engineering for embedded systems. Proceedings of the 27th International Conference on Software engineering, 2005, p. 712-713. ACM Press, USA.

[Czarnecki and Eisenecker 2005] Czarnecki, K.; Eisenecker, U.W.: Generative Programming: Methods, Tools, and Applications. 2005, Addison-Wesley Professional, USA.

[Eisenbarth and Simon 2001] Eisenbarth T., Simon D.: Guiding Feature Asset Mining for Software Product Line Development, 2001.
<http://www.plees.info/Plees01/plees01-dokumente/02-simon.pdf>.

[Gomaa 2004] Gomaa H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. 2004, Addison-Wesley Professional, USA.

[Graaf, Lormans and Toetenel 2003] Graaf, B.; Lormans, M.; Toetenel H.: Embedded Software Engineering: The State of the Practice. 2003, IEEE Software, v. 20, p. 61-69.

[Kang et al. 1990] Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Peterson S.: Feature-Oriented Domain Analysis (FODA): Feasibility Study. CMU/SEI-90-TR-21, 1990, SEI, USA.

[Kang et al.. 2005] Kang K.; Kim M.; Lee J.; Kim B.: Feature-Oriented Re-engineering of Legacy Systems into Product Line Assets: a Case Study. 2005, LNCS, v. 3714/2005, p. 45 - 56, Springer Berlin / Heidelberg.

[Krueger 1992] Krueger C. W.: Software reuse. 2002, ACM Computing Surveys, v.24(2), p. 131-183.

[Leveson and Weiss 2004] Leveson N. G., Weiss K. A.: Making Embedded Software Reuse Practical and Safe, Proceedings of the 12th International Symposium on Foundations of Software Engineering, 2004, ACM SIGSOFT '04/FSE-12, v. 29, i. 6.

[Mehta and Heineman 2002] Mehta A., Heineman G.T.: Evolving legacy system features into fine-grained components. Proceedings of the 24th International Conference on Software Engineering, Orlando, FL. 2002, ACM Press, USA. p. 417-427.

[O'Brien 2005] O'Brien, L.: Reengineering, Carnegie Mellon University Pittsburgh, USA. <http://www.cs.cmu.edu/~aldrich/courses/654-sp05/handouts/MSE-Reeng-05.pdf>.

[Riebisch et al. 2002] Riebisch, M.; Böllert, K.; Streitferdt, D.; Philippow, I.: Extending Feature Diagrams with UML Multiplicities. 6th World Conference on Integrated Design & Process Technology, 2002, USA.
<http://www.theoinf.tu-ilmenau.de/~riebisch/publ/ IDPT2002-paper.pdf>.

[Sochos, Riebisch and Philippow 2006] Sochos, P.; Riebisch, M. e Philippow I.: The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines. 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, 2006, p. 308-318.

[Szyperski 2002] Szyperski C.: Component Software: Beyond Object-Oriented Programming. 2nd Edition, 2002, Addison-Wesley, USA.

[Vahid and Givargis 2002] Vahid, F.; Givargis, T.: Embedded System Design: A Unified Hardware/Software Introduction, 2002, John Wiley & Sons, USA.

[VeriFone 2008] VERIFONE: The Way to Pay, 2008.
http://www.verifone.com

[Visanet 2008] VISANET Brasil, 2008.
http://www.visanet.com.br/VOL

[Ziadi, Jézéquel and Fondement, 2003] Ziadi, T.; Jézéquel, J-M.; Fondement, F.: Product line derivation with UML. In Proceedings Software Variability Management Workshop, University of Groningen, 2003.
<http://lglpc35.epfl.ch/lgl/members/fondement/docs/papers/Ziadi03b.pdf>