A Case Study in Software Product Lines - The Case of the Mobile Game Domain

Leandro Marques Nascimento¹, Eduardo Santana de Almeida¹,
Silvio Romero de Lemos Meira¹
¹ Federal University of Pernambuco and Recife Center for Advanced Studies and Systems - C.E.S.A.R, Recife, PE, Brazil
{lmmn2, esa2, srlm}@cin.ufpe.br

Abstract

Software Product Lines (SPL) processes are gradually being adopted by many companies in several domains. A particular domain where the adoption of such processes may bring relevant benefits is the mobile applications domain given the big diversity of handsets, although the characteristics of this domain usually create barriers to apply these processes in practice, such as, restrictions of memory size and processing power and different API implementations by different manufacturers. In this context, this work presents briefly a practical approach to implement core assets in a SPL applied to the mobile game domain combining the good practices from the already published processes and describes in details a case study performed with the application of this approach, based on three different adventure mobile games. The results of the case study have shown the approach can be suitable for the domain in question.

Keywords: software product line, case study, mobile game, core assets, variability, product map.

1. Introduction

One of the key factors for improving quality, productivity and consequently reducing costs in software development is the adoption of software reuse – the process of creating software systems from existing software rather than building them from scratch [1]. Several research activities have been done describing techniques, methods and processes in software reuse area [2] and different efforts have been made to apply the concepts of this area in practice with successful results [3], including big companies such as Hewlett-Packard and Bosch.

One approach commonly cited in the software reuse area is Software Product Line (SPL), which “is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way”, according to [4].

In addition, a particular market segment that is in high-growth during the last years is the mobile applications market, especially the mobile games. According to iSuppli¹, the prediction is that the worldwide mobile gaming market will be worth $6.1 billion in 2010 – up from $1.8 billion in 2005. However, this market involves much more complex challenges, mainly because of the variety of handsets and manufacturers, many platform restrictions and different hardware configurations. This scenario demands well defined software processes to build applications compatible with as many handsets as possible. Therefore, SPL can be a suitable option in this case.

In SPL research field, different works have been done [4, 5, 6, 7, 8, 9] applying various techniques in different contexts. Despite of these relevant works, there is still a lack of details in the phases of the SPL processes related to domain implementation, such as: how features can be mapped on components and correspondingly on code; how variability can be managed in code level; or guidelines to apply variability implementation techniques. The lack of details in domain implementation phases can make SPL approach unaffordable especially when it is taken into account the restrictions of mobile game domain, such as memory size, processing power, different screen sizes and different API implementations. Because of these characteristics involving the mentioned domain, the adoption of a SPL approach should be supported by as many details as possible at code level.

Considering these aspects, the purpose of this work is to establish a practical approach for implementing assets in a SPL applied to mobile game domain and analyze this approach through a case study involving three adventure mobile games developed with Java Micro Edition [10] technology and the production of a fourth game exploring the common features among them. This approach applies different techniques for variability implementation, providing guidelines for

¹ iSuppli Corporation: Applied Market Intelligence: http://www.isuppli.com/
mobile game developers within a SPL context, once the focus of this work remains on code level.

This paper is organized as follows: Section 2 presents the related work, highlighting their drawbacks and mentioning what this approach is supposed to cover. Section 3 discusses briefly the proposed approach. Next, the case study with a practical application of the current approach is described and analyzed in Section 4. At last, Section 5 presents the concluding remarks and future work.

2. Related Work

As it was already mentioned in the last section, several research activities have been done in the areas of software reuse and SPL. Most of the works published in these areas commonly present SPL based on the three essential activities: core asset development, product development, and management [4]. These works can be characterized as model-based processes with common concepts such as: feature modeling, domain (or product line) architecture, and component composition. Examples of these works are: Software Engineering Institute’s framework for SPL [4], PuLSE [5], KobrA [6], SPL Framework [7] from Pohl, Bockle, and van der Linden, PLUS [8].

On the other hand, other researchers take advantage of generative programming, such as FAST [9], applied in industrial cases such as Lucent Technologies and Avaya Labs.

All the works cited can be applied in different contexts, however, in the particular case of mobile domain, adaptations in the processes should be done to make them address properly the restrictions of the early cited domain. Some efforts were made to adapt a defined SPL process to mobile game domain, such as the GoPhone Project [11], an adaptation of PuLSE and KobrA. This project establishes a SPL using four target handsets with a set of four features, but it does not provide details on how to implement variability at code level and also did not consider common restrictions in the mobile domain, such as screen size and different API implementations according to different vendors.

In addition, other works define refactoring tools for extracting product lines from different versions of the same product, for example FLIP [12]. As a common practice in the mobile applications domain is to adapt an application for different handsets (porting), FLIP is intended to analyze the code of those different versions of the same application and then generate a product line based on aspect-oriented programming [13]. This kind of tool can be extremely useful when the product line is defined by a single product and different versions of it must be implemented to support different families of handsets.

However, not only is our approach intended to consider the different versions of the same product as potential variabilities to be used in other products, but it addresses the variability and commonality of different products in the same domain at a higher level of abstraction.

3. A Practical Approach for Implementing Core Assets in a SPL applied to the Mobile Game Domain

Based on the best practices of many different software reuse and SPL processes [2] in conjunction with our industry experience, we structured an approach to cover the main specific aspects of mobile applications domain, as shown in Figure 1.

The approach is focused on the domain implementation phase of a SPL, abstracting the previous phases of domain analysis and design. The process is iterative, although the Figure 1 does not show it clearly to avoid complexity and misunderstandings.

Three distinct phases can be highlighted in the process: Component Modeling, Component Implementation, and Component Testing. Each of them is summarily described on the following subsections.

![Figure 1. Process overview of SPL domain implementation phase applied to mobile domain.](image)

It is expected from the phases of domain analysis and domain design some artifacts that are used as input for the domain implementation and also throughout the process’ phases. These artifacts are:

- **Product portfolio**: describes all the families of products, highlighting the family (or base) members, which can be used as a reference for the entire family.
- **Product map**: holds information about main capabilities of products, including the restrictions inherent to the mobile domain.
• **Domain architecture**: modeled in terms of UML, the domain architecture is used to guide the development of new applications in the target domain.

• **Feature model**: described in terms of visual models following well accepted notation, such as FODA [14], this artifact shows the relationships between domain features.

### 3.1. Component Modeling

One of the main inputs for this phase is the domain architecture, which is mandatory for the appropriate process application. The main goal of this phase is to obtain the component detailed design (or component specification). For this purpose, the domain architect is the most important role, responsible for analyzing the architecture and arranging components according to the information of the product portfolio and product map, taking into account design patterns. This work does not focus on component identification based on domain architecture and feature model, but it can be observed as an important research field [15].

The product portfolio and product map are important inputs for domain architect once they may bring up some restrictions in the whole architecture, such as, a family handsets which have reduced memory size. Those documents can be constantly refined during the phases of component implementation and testing.

To start performing the component specification, the KobrA approach [6] is used as a reference and adapted. KobrA approach was chosen to be adapted because it describes a component using two levels of abstraction (internal and external point of views) and this makes it easier for developers and component users (integrators) to understand the component behavior. A KobrA component is based on specification and realization models, which are composed by a general set of UML models.

In our approach, we discourage the use of textual models, such as the decision model, to avoid misunderstandings problems with textual representations and maintenance problems if the textual model has to reflect the big number of product families. On the other hand, we encourage the use of the structural model, which is the base for component implementation, combined with variant stereotypes to indicate a variant part of the component internal design.

The domain expert may be consulted to clarify any information about the product families and their restrictions. Developers may also help on specifying components and their interfaces.

### 3.2. Component Implementation

Once the components internal design has been accomplished, the developers are responsible for implementing these components using information from component specification (structural models), product portfolio descriptions and product map constraints. Also, domain architecture can possibly be consulted, as it may be necessary to write “glue” code for the components to be integrated into architecture.

The major concern at this point is to codify the domain restrictions and variability within the components being implemented, using the already known techniques for variability implementation in product lines [16]. Among these techniques, only two of them, reflection and dynamic link libraries (DLLs), are not applicable to mobile domain because the last versions of Java profile and configuration, MIDP/CLDC, 2.1/1.1 respectively, still do not support them.

The variability implementation techniques that are applicable to mobile applications domain are: conditional compilation, aggregation / delegation, inheritance, parameterization, overloading, dynamic class loading, properties, static libraries, aspect-oriented programming (AOP), and design patterns. During the execution of case study in Section 4, some practical examples are provided to help developers during component implementation for the applicable techniques.

### 3.3. Component Testing

Once the components have been implemented, they must be tested to verify if they are working properly. The scope of these tests are much more related to unit tests, once a component certification process is the most appropriate way to ensure component quality [17]. The main inputs for this phase are: the components code (documentation included), example applications used to integrate and test those components and a test plan.

An example application can be either an instantiated product of the SPL or a simple application developed to test component basic functionality. This information about how a specific component is going to be tested should be provided in test plan.

The test plan should be structured based on the information of product portfolio and product map. The tests should be focused on the base members of the families, once it is not viable to test all the members of all families. It is expected that if a component is working fine in the base member of the family, it will work properly on the other members of that family. At least, it is what is expected from the handset manufacturers.

In addition, the test plan should be divided in two parts: the domain test plan, which explores the commonalities among the features and may be used across different product families in the product line; and the specific product test plan, which explores the handset specific features.
4. The Case Study

In order to evaluate the described approach, we performed a case study following the organization proposed in [18], which is composed of five phases: definition, planning, operation, analysis and interpretation. Sometimes the future tense is used, symbolizing the precedence of the plan in relation to its execution.

4.1. The Definition

For the definition of the case study, the GQM paradigm [19] was used. The GQM is based upon the assumption that for an organization to measure in a purposeful way it must first specify the goals for itself and its projects, then it must trace those goals to the data that are intended to define those goals operationally, and finally provide a framework for interpreting the data with respect to the stated goals.

4.1.1. Goal

G1. To evaluate the approach for core assets implementation in a SPL applied to mobile applications domain in terms of efficiency and appropriate management of domain restrictions, including traceability from domain features to code level.

4.1.2. Questions

Q1. Does the approach generate components with low complexity?
Q2. Does the approach manage the domain restrictions, such as application size, different screen sizes and different API implementations?
Q3. Can the approach map domain features to code level?

4.1.2. Metrics

M1. Component Complexity. We defined component complexity metric in terms of cyclomatic complexity, which is a mathematical technique developed by McCabe [20] and provides a quantitative basis for modularization, identifying software modules that will be difficult to test or maintain. Cyclomatic complexity is measured in terms of the number of paths through a program. This measure is calculated from a connected graph of the module (or component):

\[
CC(G) = E - N + p,
\]

where:
- \( E \) = the number of edges of the graph
- \( N \) = the number of nodes of the graph
- \( p \) = the number of connected components

The proper interpretation of this metric, based on research performed in the Software Engineering Institute (SEI), is based on four different intervals for the values: 1 and 10, it is a simple program, without much risk; between 11 and 20, it represents a more complex program, with moderate risk; between 21 and 50 represent, it is a complex program, with high risk; and greater than 50, which represents a program impossible to be tested1 and presents a high risk.

M2. Domain Restrictions Management. As it was already mentioned, the mobile applications domain has specific characteristics different from other domains, which may restrict the use of a SPL approach. In order to make feasible the application of such approach, the restrictions related to application size, different screen sizes and different API implementations, for example, must be considered. In this context, three metrics related to restrictions managements can be used:

\( NF_A \): %Product families which have the restriction of application size mapped at code level.
\( NF_S \): %Product families which have the restriction of different screen sizes mapped at code level.
\( NF_I \): %Product families which have the restriction of different API implementations mapped at code level.

As these metrics have never been used before, there are no well known values for them. The practical experience and common sense were used to define these values. Thus, a percentage above 50% for all three metrics is considered as an indicative that the approach can manage the mobile applications restrictions.

M3. Traceability Level. One of the main concerns related to software processes is the traceability among artifacts. In practice, however, some artifacts may not be traceable. For example, a component that presents three different variabilities modeled in its specification, but these variabilities can not be traced to code level. In order to avoid this situation, we define two metrics for better managing the traceability among artifacts:

\( TCF \): %Components that can be mapped to domain features.
\( TCC \): %Components that can be mapped to code (respective conditional compilation tags).

Also based on practical experience, as cited in metric \( M_3 \), an acceptable value for the two metrics of traceability level is a value above 70%.

4.2. The Planning

After the definition of the case study, the planning is started.

Context. The objective of the case study is to evaluate the SPL approach in a mobile game project. The project will be conducted with one software engineer who has previous experience in mobile games.

Pilot Project. Before performing the study, a pilot project will be conducted with the same structure
defined in this planning. The pilot project will be performed by a single subject.

Subjects. The subjects of the study will act as domain experts, domain architects, software engineers, reuse consultant and designers.

Criteria. As the study is focused on quality, some criteria is needed to evaluate the benefits of the approach. The benefits will be evaluated quantitatively in terms of components complexity, using McCabe cyclomatic complexity, domain restrictions management and traceability, by observing product map and portfolio and its correspondence to implemented code.

Null Hypotheses. These are the hypotheses that the experimenter wants to reject with as high significance as possible. Therefore, according to the selected criteria, the following hypotheses can be defined:

- H₀: µthe approach generates components with CC >= 21
- H₀: µNFA <= 50%
- H₀: µNFS <= 50%
- H₀: µNFI <= 50%
- H₀: µTCF <= 70%
- H₀: µTCC <= 70%

Alternative Hypotheses. These are the hypotheses in favor of which the null hypotheses are rejected. Thus, the following hypotheses can be defined:

- Hₐ: µthe approach generates components with CC < 21
- Hₐ: µNFA > 50%
- Hₐ: µNFS > 50%
- Hₐ: µNFI > 50%
- Hₐ: µTCF > 70%
- Hₐ: µTCC > 70%

Independent variables. The independent variable considered in this case study is the expertise of the subjects with the mobile applications domain.

Dependent variables. The dependent variables are the quality of the components and product map/portfolio information and the number of product families. The components quality will be measured by their complexity (McCabe cyclomatic complexity). The product map/portfolio quality will be measured by the traceability level and the number of families will be measured in terms of domain restrictions management.

4.3. The Project used in the Study

We used three games of the same domain from a mobile game producer¹: Monga, American Dad – Roger’s Scape and Zaak. Figure 2 shows a screenshot of each game. Based on the features and consequently code from the early cited games, this case study aims to establish a SPL, using the approach described by this paper, and produce a fourth game from this SPL.

The first game mentioned – Monga – is a platform based game where player is a gorilla, called Monga, who loves his audience and wants to catch them. All the people fearing the gorilla start to run through the floors of the building and the player’s objective is to conduct Monga to catch the audience. During the running, the gorilla faces some obstacles in order to difficult the audience catching.

In the second game cited – American Dad - Roger’s Scape – the player is an alien walking around a simulated labyrinth and facing humans prepared to follow and catch him. These human enemies have a kind of artificial intelligence and start to chase the player when he enters in their perception range. The main objective of the game is to find the correct keys through the labyrinth and open the respective doors to find a way out to the player’s original world. The player can attack enemies by squirting a liquid on the floor which grabs the enemies making them immobile for few seconds.

The last game in the sequence – Zaak – is a platform based game where the player is in charge of guiding little creatures (fairies) through a world full of obstacles to an exit. During the guidance, he can face enemies that keep on walking around the level in a predefined path (patroller) and can attack and destroy them using magic cards. Through the levels, the player can collect different items that help on guiding the fairies to the way out, such as ropes and leverages.

![Figure 2. Screenshots of the games in case study. A) Monga. B) American Dad – Roger’s Scape. C) Zaak.](image)

The subjects involved in the project had full access to documentation and code. However, not so many details of the games will be provided because of matters of confidentiality.

4.4. The Operation

Experimental Environment. The study was conducted as part of Nascimento’s (first author) M.Sc. degree, during December 2007-March 2008, at Federal University of Pernambuco, Brazil.

Subjects. The subjects involved were one M.Sc. student and his co-advisor, both from Federal University of Pernambuco.

Summary of Generated Artifacts. Following the early described approach and the suggested sequence of steps, the main activities during the execution of case study were: domain analysis, resulting in a domain feature model; definition of product map and

product portfolio; architecture definition; component implementation and testing; and instantiation of a product based on the defined components.

Firstly, a feature model was defined, capturing the commonalities and variabilities among the three games. The commonalities are related to the features Audio, Menu and Player Actions. On the other hand, the variabilities are related to Enemy and Scenario. Monga does not involve enemies directly, but only moveable obstacles in the scenario. The other two games have two different types of enemies, one for each game: the first type is the Chaser, which chases the player around the scenario; the second type is the Patroller, which walks around the scenario in a predefined path. Concerning Scenario feature, it can be observed two different types of scenario for the three games: Plane (American Dad) and Platform (Monga and Zaak).

After structuring the feature model, it is time to define product portfolio and product map. The former describes all the families in details, highlighting each peculiarity of them. The later maps each capability of product families to conditional compilation tags. Table 1 shows an example of a product map.

Table 1. Product map in a reduced form with conditional compilation tags attached.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Variation Level #1</th>
<th>Variation Level #2</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen Size</td>
<td>130x130</td>
<td>scr_130x130</td>
<td></td>
</tr>
<tr>
<td></td>
<td>176x220</td>
<td>scr_176x220</td>
<td></td>
</tr>
<tr>
<td></td>
<td>240x320</td>
<td>scr_240x320</td>
<td></td>
</tr>
<tr>
<td>Key Mapping</td>
<td>Nokia</td>
<td>keys_nokia</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Motorola</td>
<td>keys_mot</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SonyEricsson</td>
<td>keys_sony</td>
<td></td>
</tr>
<tr>
<td>Sound</td>
<td>Allocation Mode</td>
<td>Pre-allocate</td>
<td>snd_prealloc</td>
</tr>
<tr>
<td></td>
<td>On demand</td>
<td>snd_demand</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Player interface</td>
<td>Block</td>
<td>snd_block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Thread</td>
<td>snd_thread</td>
</tr>
<tr>
<td>Known issues</td>
<td>Network</td>
<td>UDP problem</td>
<td>issue_net</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>No garbage</td>
<td>issue_mem</td>
</tr>
</tbody>
</table>

At this point, the domain architecture should be modeled. As it is not the focus of this SPL approach being followed and for matters of confidentiality, domain architecture is not going to be detailed.

When instantiating domain architecture, the architect must decide whether these classes are going to be attached to the new product in the product line by copying and pasting the code or by configuring a library (JAR file) in the project. The first option is only indicated when there are very restrict families in terms of application size (final JAR file size).

Once the architecture, product portfolio and product map are done, the components can be specified, using variant stereotypes. Four components have been identified: Sound, Main Menu, Player and Enemy (AI).

At this point, the next steps involve developing the defined components using a subset or combination of the variability implementation techniques [16]. Following is an example code snippet involving the Main Menu component cited. Note that the entire Animation class is embraced by conditional compilation tags and it would not even exist if the feature is not defined.

```java
/**
 * public class Animation {
 *     void play(){
 *         //ifdef issue_mem
 *         this.playShortAnim();
 *         //else
 *         this.playLongAnim();
 *         //endif
 *     }
 *     void playShortAnim(){/**/}
 *     void playLongAnim(){/**/}
 * }
 */
```

Another example of combination of inheritance and conditional compilation is shown in the following code snippet. Note that the entire subclass has to be embraced by conditional compilation tags in order to reduce final application size.

```java
public abstract class Enemy {
    public abstract void walk();
}
//ifdef scr_176x220
public class Chaser extends Enemy {
    public void walk() {/*Chase player*/
    }
    //endif

//ifdef scr_130x130
public class Patroller extends Enemy{
    public void walk() {/*Walk around*/
    }
    //endif
```

As the focus of this paper is to describe the case study, no more guidelines or code snippets are going to be presented.

With the components defined and implemented, the test team must use an example application to test each component considering the families described in product portfolio. Each component should be tested in a representative member of the family as it would be very time consuming to test every handset in the family.

The last step is to perform a typical application engineering phase described in details by different authors, but it is not the focus of this work.

At the end, four components were developed, as shown in Table 2. The basic domain architecture has been implemented with 6489 lines of non-commented source code distributed in 277 methods.
Table 2. Components overview.

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines of non-commented source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound</td>
<td>114</td>
</tr>
<tr>
<td>Main Menu</td>
<td>661</td>
</tr>
<tr>
<td>Player</td>
<td>383</td>
</tr>
<tr>
<td>Enemy (AI)</td>
<td>417</td>
</tr>
</tbody>
</table>

4.5. The Analysis and Interpretation

After operation phase was executed and based on the project information, metrics could be collected. Then, the null hypotheses described in the early sections can be rejected.

Component Complexity. Table 3 summarizes the values for component complexity (cyclomatic complexity) that were obtained. The component complexity mean (11.2) rejects the null hypothesis. It indicates that the proposed approach aid on generating components with low complexity. Even the highest complexity obtained (17.1) for the Enemy (AI) component presents the value below the null hypothesis.

Table 3. Component complexity values.

<table>
<thead>
<tr>
<th>Components</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound</td>
<td>7.6</td>
</tr>
<tr>
<td>Main Menu</td>
<td>6.8</td>
</tr>
<tr>
<td>Player</td>
<td>13.4</td>
</tr>
<tr>
<td>Enemy (AI)</td>
<td>17.1</td>
</tr>
<tr>
<td>Mean</td>
<td>11.2</td>
</tr>
<tr>
<td>Null Hypothesis</td>
<td>\geq 21</td>
</tr>
</tbody>
</table>

Domain Restrictions Management. Table 4 summarizes the values for the metrics defined in Section 4.1.2. All three values related to domain restrictions (application size, different screen sizes and different API implementations) reject the null hypotheses. It is important to highlight that these values may not be much accurate. It may be necessary to perform new studies for better calibrating these values considering the specific characteristics of the domain in use. The hypotheses rejection indicates that the proposed approach can handle the restrictions related to mobile applications domain. However, the practical experience shows that if there are many different families (we worked with four), these restrictions may become harder to manage.

Table 4. Domain restrictions management values.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFA</td>
<td>75</td>
</tr>
<tr>
<td>Null hypothesis (NFA)</td>
<td>\leq 50</td>
</tr>
<tr>
<td>NF5</td>
<td>100</td>
</tr>
<tr>
<td>Null hypothesis (NF5)</td>
<td>\leq 50</td>
</tr>
<tr>
<td>NF</td>
<td>75</td>
</tr>
<tr>
<td>Null hypothesis (NF)</td>
<td>\leq 50</td>
</tr>
</tbody>
</table>

Traceability. Table 5 summarizes the values related to traceability from components to features and to code. Both values reject null hypotheses indicating that the proposed approach can manage traceability within a mobile game SPL. These values where obtained through observations of domain feature model, components structural model, product map and components code.

Table 5. Traceability values.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCF</td>
<td>75</td>
</tr>
<tr>
<td>TCC</td>
<td>100</td>
</tr>
<tr>
<td>Null hypothesis (TCF)</td>
<td>\leq 70</td>
</tr>
<tr>
<td>Null hypothesis (TCC)</td>
<td>\leq 70</td>
</tr>
</tbody>
</table>

4.6. The Lessons Learned

One of the strongest points to be highlighted in this project is related to the facility of applying conditional compilation tags in code, mainly when the product map explicitly maintains all information about handsets capabilities and their respective tags. On the other hand, the use of these tags can become problematic if the number of product families grows too much.

Considering the aspects related to domain restrictions management, we realized that it may be necessary to perform new studies to calibrate the value of null hypothesis for component complexity metric.

In addition, as a lesson learned for this project, it can be mentioned that it is unavoidable the use of a tool to manage all conditional compilation tags, mainly when the number of product families becomes high.

5. Conclusions and Further Work

Software product lines are being explored in different domains and contexts with many successful cases. Moreover, a market segment that is drawing attention of industry and academia, because of its high-growth is the mobile game domain. This domain presents specific characteristics, especially because of the great diversity of handsets and the need of ubiquitous applications running in as many handsets as possible. Thus, some efforts have been made to apply SPL approach to mobile game domain. However, the current work in SPL area does not address properly the mentioned characteristics of the domain, mainly in domain implementation phase. In this paper, we described a practical approach to develop core assets in a SPL providing details at code level and then performed a case study with three mobile games following the mentioned approach and using the model proposed by [18]. The results of the case study have shown that the approach can be suitable for the mobile domain.

As future work, we are planning to apply this approach in an industrial context and evaluate the benefits in a 3D game project with 8 people involved.
The team has previously developed two different games in the same domain, which will be used as the base to apply the approach described by this work and produce a third game, using 3D capabilities.

Acknowledgments

We would like to thank Dr. Dirk Muthig from Fraunhofer Institute and Dr. Bill Frakes from Virginia Tech for important discussions about software product lines, which contributed for this paper. We also would like to thank Meantime Mobile Creations for providing all the artifacts related to the games used in the case study of this work and supporting us with technical experience in the game development field. This work is sponsored by Brazilian Agency (CNPq process number: 475743/2007-5).

References