The Mobile Product Line

LUKA PAVLIČ, University of Maribor

In this paper we present the challenges during the simultaneous development of multiple mobile application versions with different sets of functionalities. Some of them are core, the other optional, and the third alternative. One of the indicated appropriate solutions is the approach of software product lines. In this paper we presents practical experiences during the product line implementation in the case of mobile applications for the Android operating system. It has at least six to eight different versions simultaneously available. Among others, these are freely available version, paid version, development, test and demonstration version. We also offer certain versions for the BlackBerry phones, some mobile application components share common functionalities with a portal server.

General Terms: Software Engineering

Additional Key Words and Phrases: Software product lines, Mobile applications, Mobile product lines, Android

1. INTRODUCTION

Software development has always been pursuing some universal business goals in the context of the business environment. These include high-quality software, rapid development, fast market penetration, low development costs, low maintenance cost, the possibility of rapid changes and so on. Often we are faced with software development projects, where the product is just one piece of specialized software. Such project would definitely not pursue above mentioned business goals sufficiently. In the scope of software reuse, software engineering has already given some answers on how to enable rapid development, low cost and high quality at the same time. One of these answers can be found in the software product lines (SPL). In this paper we will present practical experience with the introduction and successful implementation of the SPL approach in mobile solution @life (www.a-life.eu.com). In the second section we present the theoretical foundations of the SPL in terms of motivation, attitudes and acquisitions. The third section gives some practical aspect of the SPL for mobile applications for the Android operating system.

2. SOFTWARE PRODUCT LINES

2.1 Motivation: Gains from the Reuse Point of View

Reuse as one of the fundamental disciplines of software engineering plays an important role in the development of new, or maintenance of existing systems. It is almost impossible to have a software, that would result in a single version. One of the typical example of a well-established and diversified versions by the same piece of software are widely available commercial packages that offer a different set of functionalities according to the users' needs and/or financial investment. E.g. publishing software packages are available to private users, entrepreneurs or large organizations. Software could also be tailored to a specific set of hardware. After all, we are talking about diversity even when we have at the declarative level only one version of software. Even in this case we still need to have test environment software, production environment software etc.

These challenges are addressed with reuse approach. The reuse in software engineering has always dictated development models. From modular development in the sixties and seventies, through the object, component and later service-oriented development.

While developing a set of similar software products, separated by only a certain small set of specialized functionality for individual products, with the possibility of different implementations of the same

Author's address: L. Pavlič, Institute of informatics, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova ulica 17, 2000 Maribor, Slovenia; email: luka.pavlic@uni-mb.si.

In: Z. Budimac (ed.): Proceedings of the 2nd Workshop of Software Quality Analysis, Monitoring, Improvement and Applications (SQAMIA), Novi Sad, Serbia, 15.-17.9.2013, published at http://ceur-ws.org

functionality, we can do this without the approaches of reuse, or just with the appropriate use of established reuse methods: component development, patterns, libraries etc. SPL in addition to the existing mechanisms of reuse allow some other levels of reuse - reuse at the level of larger software pieces. Besides reusing technical building blocks, these include also reuse of the procedures and rules, associated with the software. They include single analytics, planning and management of software development. This is how SPL enables lowering development costs and raise of quality at the same time. SPL approach could be implemented when some of the following issues occur as a result of the complexity of the software [Northrop, 2013]:

- we develop the same functionality for a variety of products and/or customers,
- the same change should be made in a number of different software products,
- he same functionality should behave differently depending on the final product,
- certain functionality can no longer be maintained, and so the customer has to move to newer version of the software,
- we can not estimate the cost of transferring certain features to another software,
- certain basic infrastructure changes lead to unpredictable behavior of dependent products,
- the majority of effort is put into maintenance, and not the development of new functionality.

At this point, we can draw a parallel with other engineering disciplines, where such approach (product lines) are not only present, but also fully implemented. E.g. the automotive industry: based on the same chassis manufacturers produce a range of different vehicles within a common family. Certain vehicles may have a transmission made in the version of the automatic gearbox, certain vehicles can have the accelerator pad and the brakes also on the passenger side (for driving schools), the third type of the same family of vehicles offers a rather large backpack, third row of seats etc. The chassis of all these vehicles is not developed with the idea of completely universal, reusable element, but as a basis for known variability of vehicles. Thus, for example, the chassis do not offer the possibility of two pairs of rear wheels, as it is already clear from the beginning that the chassis was developed for four-wheeled vehicle. In addition to cost-effectiveness, such approach helps to achieve higher and, more importantly, more manageable quality. You can also have a separate test line, where you can develop improvements independently with possibility of introduction in production later. When, for example, improving rear door locking mechanism, the improvement may automatically reflected on all vehicles with rear doors, regardless of whether it is a commercial vehicle for the driving school, etc.. without any influence to the process of manufacturing or even undermine the basic plan family cars.

An example of this can be directly brought on the software using a software product lines. Software product lines mean just that: a common basis that share entire family of products. In doing so, we want to have also common support and development activities, such as planning, quality control, production engineering packages, unified debugging, etc..

One of the definitions of software product lines is as follows [SEI, 2013]:

A software product line (SPL) 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.

Software product lines provide on the one hand lowering the development costs, and on the other hand higher quality, both because of well-known common points of product family. Such an approach requires the development of appropriate architecture with possibility of introducing variabilities per product.

Nevertheless, the product line also require additional costs: architecture, building blocks and individual tests should include the possibility of variability, business plans must be made for multiple products, not just one, etc. However, in the long term the contributions of software product lines proved to be as follows [Northrop, 2013]:

- up to 10x improved productivity,
- up to 10x improved quality,
- joint development costs reduced up to 60%,
- shortened time-to-market up to 98%,
- the possibility of moving to new markets is measured in months, not in years.

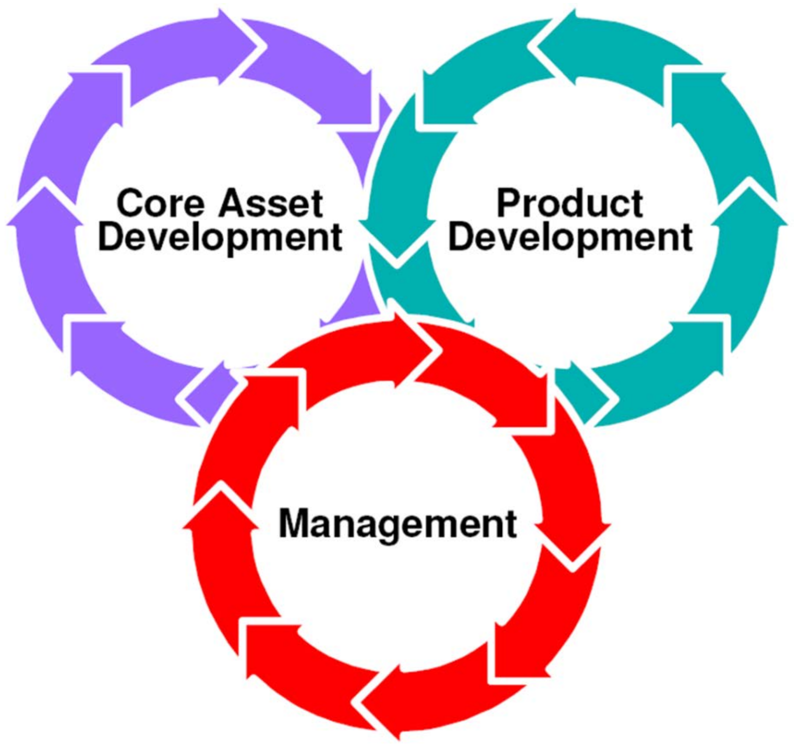

Fig. 1. Development activities, according SPL [Northrop, 2002]

As shown in Figure 1, the SPL approach separates the development of basic building blocks and their assembly into finished products [Northrop, 2002]. Within the development iterations we have to combine separate building blocks into product. The development of the basic product is thus completely separated from the assembly, while the assembly into the final product is completely automated. Management activities include supporting technical and organizational tasks.

It is also important to clarify, for which examples SPLs are not suitable. As a product line approach cannot be understood [Muthig, 2004]:

- reuse in general, typical of small general purpose building blocks,
- development of a unified system through reuse (libraries, components, services, samples, etc..),
- classic component or service-oriented development,
- architecture with the possibility of configurable behavior,
- versions of certain software.

2.2 Variability in Software Product Lines

Different authors give different definitions of variability. One of the most easy to read goes as follows:

Variability is the ability of software to be effectively extended, modified or adapted to use in a particular context. [Cavalcanti, 2013]

Keeping variation (difference or equality) in the individual building blocks of the final products is the basis of software product lines. In addition to the presence or absence of a functionality in a particular line we also often have specific functionality realized differently according to the line. These differences are recorded in the modeling process variability, which is an integral part of the system analysis. During the analysis we identify variable points, which later results in different realization of building block and final software:

- Functionality presence: If the functionality is present in all the lines and in all with the same realization, such functionality may be realized in the most general common building block.
- The lack of functionality: the functionality is not present in particular lines. In the case that the functionality is required in only one line, the functionality may be realized which in the line itself, otherwise it is necessary to introduce specific building block.

- A different realization: the functionality is available, but the realization will be different in different product lines. Different realization can be realized in line, unless the same feature can be found in multiple lines - in this case, it is reasonable to introduce a new building block, which is a specialization of the existing one.

Variability should be taken into account at all levels of analysis, design and development. In addition, it is necessary to pay special attention to the variability even during development in the context of management activities. Identification of variability is one of the main challenges of product lines. Other challenges are related to the realization of variability, their inclusion in the product line and adequately control variability. In addition, the possibility of variation functionalities (identified the variability in the level of functionality) on the technical level are mapped to a number of possible types of variability: [Muthig, 2004]

- variability of the data,
- variability in the level of implementation applications,
- variability at the level of the technology (software or hardware),
- variability in the level of quality criteria,
- variability of the target environment.

The variability technical realization is based on already established and well-known concepts in software engineering. To name a few [Clements, 2005]:

- building block inclusion, e.g. components, services,
- use of design patterns (factory, abstract factory, bridge, etc),
- changing behavior through inheritance,
- changing behavior with plugins,
- parameterization,
- configuration with deployment descriptors,
- directives at compile time,
- generating source code.

The best technical solution for realizing of variability will certainly contain proven and standard procedures in software engineering. It will also provide long-term stability in means of development and maintenance.

3. PRACTICAL EXPERIENCES – SOFTWARE PRODUCT LINE FOR ANDROID APPLICATION

3.1 @life Mobile Software Product Line

@life (www.a-life.eu.com) is a solution for people who need targeted support in detecting, monitoring and eliminating the negative effects of stress. It is intended for individuals in strengthening and upgrading health reserves and as such focuses on a healthy lifestyle.

Within the @life ecosystem, we have web, mobile, tablet and desktop applications. They are integrated with the @life cloud.

The mobile application has relatively large number of functionalities. They include:

- stress-assessment questionnaire,
- guiding physical tests,
- manage measurements, such as measurements of weight, blood sugar, blood pressure, etc..,
- perform advanced measurements of heart rate,
- guided exercises for strength, flexibility, balance, supported by videos and instructions,
- guided psychological exercises, such as relaxation, autogenic training, breathing exercises, etc..,
- performing outdoor activities and record various parameters such as heart rate, distance, altitude, calories consumption, speed, etc..,
- guided activities, such as interval training,
- activities diary, supported by analyzes in the form of graphs and geolocation services,
- measurements diary in the context of advice, based on deviations according to the desired value,
- guided psychological, physical or medical programs,
- etc.

Different set of features and a different behavior of the same functionality for different users was key to development method selection: software product lines. Thus, current application at the time of selection has become the basic building block upon which the rest of the line. Current product lines are visible in Figure 2

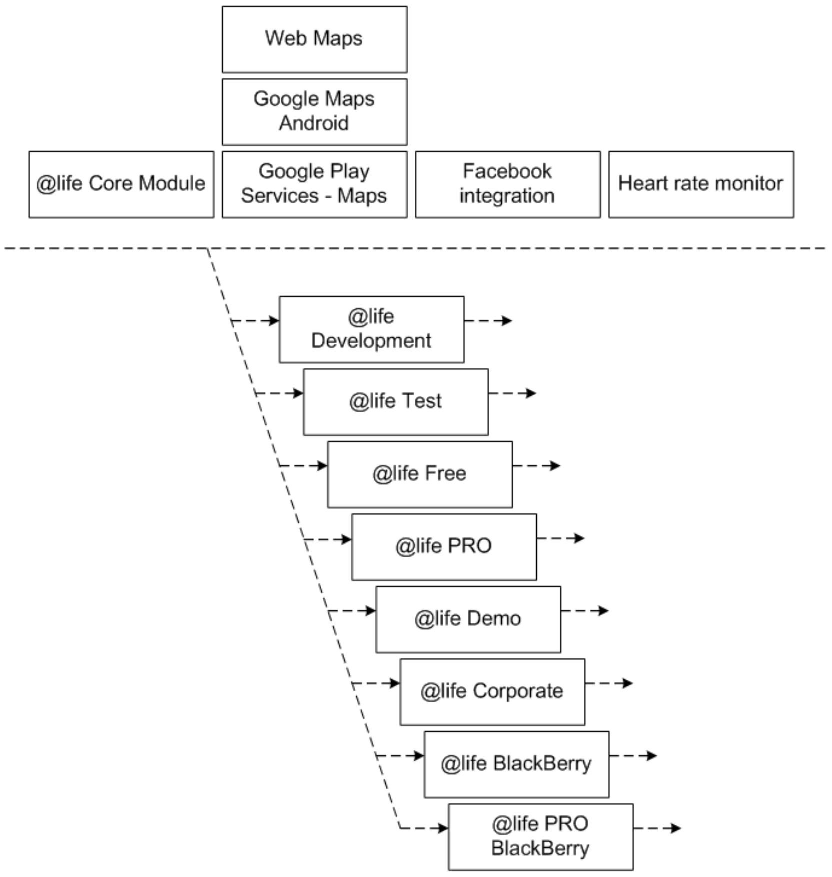

Fig. 2. @life product lines

The basic @life building block is fully running Android application. It contains only standard-based components (Android API). It realizes the functionalities that are common to all lines. At the same time, this building block also contains common architecture of mobile applications and the variability points of additional functionalities. All the basic building blocks and lines are fully operational applications - in short, it is not a library or component in the classic sense of the word. Applications within the product line are built into the final products with standard tools and Ant scripts.

Let us discuss some examples of optional and alternative features from both conceptual as well as technical point of view. One of the most illustrative examples of alternative functionality is the map that the BlackBerry phones is not supported in the form of a Google map.

Examples of optional features are:

- programs (not available in the free version),
- reminders (not available in the free version),
- exercises, videos (not available in the free version),
- demonstration and test versions expire after one month of production,
- etc.

Here are also many alternative functionalities. Let us mention only a few of the most illustrative examples:

- during the synchronization with the @life cloud, free versions do not include psychological and kinesiology exercises, nor anti-stress programs,

- test production line is connected to the test cloud, while the rest (except Corporate) is connected to the production cloud,
- current test version shows the maps using 3D acceleration eith the possibility of rotating the map, etc.., The production version contains the first generation of Google Maps, BlackBerry line offer only online map views,
- pro versions allow additional settings, e. g. fine tuning the automatic pause during activity.

There are variabilities also from language point of view. All applications are supporting six different languages. Currently we have also two more (demo) product line - one with the Chinese language support and one in Russian.

3.2 Practical examples of dealing with variabilities

The introduction of software product lines in the development of mobile solutions for @life was a challenge both from an organizational as well as technical point of view. During requirements gathering, designing and testing, functionalities were collected in a multi-dimensional table. That it was right decision also from technical aspect was proved especially because of several example situations:

- debugging and repairing found bugs,
- introduction of new product lines,
- inserting multilingual translations for several product lines.

When a quality assurance team found certain inconsistency or conflict with the specifications we had to fix it. In the case that there is an error in shared functionality, fixing error once automatically meant fixing error in all connected lines. Because of correct product line implementation we also did not expect any side effects while fixing errors, e.g. introducing new errors while fixing existing ones. The product lines has actually shorten the time of development, maintenance and troubleshooting.

Properly established architecture and software product line also allows virtually instant creation of new lines. We have to put a lot of effort to establish the appropriate architecture and variability points, which was SPL disadvantage. But in our case, this effort has repeatedly returned.

In terms of technical solutions for variabilities, we used the well-known best practices and approaches in software engineering area. These include the use of inheritance, extensions, and the component parameterization. Design patterns are also used heavily: e.g. factory, abstract factory, factory method, bridge, bean, etc..

We have used all three possibilities of variabilities in our product lines:

- include special features,
- remove unwanted features,
- changing behavior.

The inclusion of specialized functionality in the individual lines was achieved in several ways:

- preparation of the expansion point in the basic application (abstract methods, which expect concrete implementations in product lines - e.g. geolocation services),
- inheritance of existing classes and adding new methods calls in product line (e.g. calendar with reminders),
- using abstract factory pattern, which combines the functionality of the new line and its own user interface.

Exclusion of unwanted features was achieved mainly through inheritance and exclusion of unwanted features (such as not downloading programs for free lines), as well with the parameterization of the basic building blocks.

Changing behavior (e.g. line demo expires one month after construction), were achieved by appropriate design patterns, such as a bridge, factory method, builder, etc..

4. CONCLUSION

In this paper we reviewed the opportunities offered by the approach of software product lines. They were also presented in a case study on a specific project, the mobile solutions of @life. According to our own experiences, we can say that the described approach was correct and its positive effects are shown on a daily basis. We can confirm that the correct introduction of software product lines not only reduce development costs, but also significantly raise the quality. However, prior to the introduction of such approach it is necessary to keep in mind that SPL not only mean technical changes but also and mainly organizational changes and changes throughout the whole software development cycle.

REFERENCES

Software Engineering Institute, Software Product Lines, http://www.sei.cmu.edu/productlines, last visit may 2013.

Northrop, L. M. (2002). Sei's software product line tenets, IEEE Software 19(4): 32–40.

Northrop, L. M. Software Product Lines Essentials. Software Engineering Institute, http://www.sei.cmu.edu/productlines, nazadnje obiskano maj 2013.

Cavalcanti, Y. C., Machado, I. C., Anselmo, P. Handling Variability and Traceability over SPL Disciplines, Software product line – Advanced topic, Edited by Abdelrahman Osman Elfaki, Rijeka, 2013.

Clements, P. C., Bachmann, F., Variability in Software Product Lines, Product Line Practice Initiative, 2005.

Dirk Muthig, D. et. al., GoPhone - A Software Product Line in the Mobile Phone Domain, Fraunhofer Institut, Experimentelles Software Engineering, 2004.