Managing Variability throughout the Software Development Lifecycle

Neil Loughran and Awais Rashid

Computing Department, InfoLab 21, South Drive Lancaster University, Lancaster LA1 4WA, UK

(loughran | awais) @comp.lancs.ac.uk

Abstract. An increasingly important attribute of modern software development is that of *variability*. Variability allows software artefacts to be reused and configured to different contexts thus easing development, cutting costs and decreasing time to market. Most literature with regard to variability has tended to be confined to code level, however, variability can be applied and occurs at all levels of the software development lifecycle, especially when new requirements arise. We believe that variability mechanisms can be utilised at other stages in the lifecycle and non-code artefacts such as requirements, designs, tests, documentation and user manuals. In this paper we discuss and reason about how variability across the lifecycle might be supported in order to provide a holistic architectural multi-dimensional approach to software development.

1. Introduction

The ability for software to be varied to different requirements is a key property in modern software development and, in particular, software product lines. Product line architectures effectively allow the automation of software development as opposed to the creation of custom 'one of a kind' software from scratch. By utilising variability techniques, highly reusable code libraries and components can be created, thus simplifying software development, cutting costs and decreasing time to market.

Most literature on software variability tends to concentrate on the design and code level. However, variability tends to crosscut multiple points in code as well as different levels of the software development lifecycle. Moreover, the effects of variability and, in particular, new variabilities brought on by evolution, tend to propagate in ways that can not be easily modelled or managed. New requirements may necessitate changes to code, designs, tests, documentation and user manuals amongst many other artefacts and assets that go into making a product line. Ensuring the traceability of requirements and their variations throughout the software life cycle is key for a successful product line.

The management of the various artefacts in a software system are normally handled by configuration management (CM). However traditional CM practices tend to manage *variations in time* (different versions of the software system) as opposed to *variations in space* (how individual products in a product line differentiate from one another). Moreover, complex interactions between time and space dimensions can hamper management and development of a product line architecture [1].

In this position paper, we discuss how variability relates to the different phases of the software lifecycle and offer some suggestions as to how this variability can be managed and contained. We believe that treating variability as an 'aspect' that cuts across artefacts across the software life cycle can contribute to more customisable, evolvable and flexible product line architectures.

2. Variability Across the Lifecycle

Variability has been defined as "the ability of a software artefact to be changed or customised to be used in multiple contexts" [2]. The word 'artefact' is highly significant, as it does not restrict our scope to meaning just simply code level artefacts. Each stage of the lifecycle can be subject to different kinds of variability and, in the interests of process visibility, may have any number of documents and reports associated with it:

- *Variability in requirements.* Variability occurs at the requirements stage when requirements are volatile by nature (e.g., those pertaining to domains such as banking, e-commerce and telecommunications) or when requirements are not fully understood and are therefore delayed until later. Reusability of requirements is an area that has garnered great interest recently [3].
- *Variability in analysis.* The analysis stage involves analysing the requirements and performing variability and commonality analysis of the product domain. Different products in the product line may need to be treated differently when conducting early trade-off analysis, e.g., during requirements engineering or architecture design – trade-offs between concerns *A* and *B* might be more significant for product *X* while those between concerns *B* and *C* might be more critical for product *Y*.
- *Variability in design.* Variability modelling via Feature-oriented Domain Analysis (FODA) [4] and other related techniques, such as applying variability to UML models are discussed in [5]. Different, yet overlapping, sets of design elements would constitute products in the product line.
- *Variability in implementation.* Variability techniques for increasing the reusability and configurability of code such as inheritance, generics, conditional compilation, patterns and so forth are well known to developers. Each mechanism has its strengths and weaknesses [6]. Therefore, choosing the correct implementation mechanism is vitally important [7].
- *Variability in testing.* Test code can also benefit from being generalised so that they can be reused in other contexts.

- *Variability of binding time.* Different types of features and variations may require different bind times. Compile-time binding is still the predominant technique although it is not as flexible as requirements must be known in advance and hard coded. Run-time binding allows for software to bind software artefacts in a context sensitive way. The ability for a software system to be modified at run-time without having to close it down is gaining a lot of interest from researchers. Other approaches such as [8] investigate a user initiated variability approach where the system is customised at run time by the end user.
- *Variability in evolution.* It has been estimated that up to 80% of lifetime expenditure on a software system is spent on the activity of evolution [10]. Evolution is a problem for software developers, as new requirements can cause restructuring of the system and require that various documents, reports and user manuals be updated. The problem is further exacerbated in product lines as new requirements may not be compatible with all products in the domain. Handling anticipated variants can be done by leaving hooks in the software architecture or designing software with the evolutions in mind. However, unanticipated variants can cause big problems with system structure and modularity.

As previously mentioned, each phase will have its own internal documentation, such requirements documents, UML designs, code print outs, etc., as well as external documentation such as user manuals, installation procedures, training videos and so forth. As changes happen to the system or product line, these documents will have to be updated. While approaches such as the Concern Manipulation Environment (CME) [9] advocate a need for crosscutting tool support for all levels of the lifecycle, little is known about the relationships between the different levels themselves.

The underlying question is:

> *Can we capture variability, change, and its consequences as a crosscutting concern throughout the lifecycle and software architecture?*

An effective and holistic management of variability requires that all the variability dimensions must be treated uniformly across the software lifecycle with effective traceability maintained between variations at higher abstraction levels and those at implementation level as well as supporting artefacts such as documentation and manuals. Since such a treatment cuts across artefacts across the life cycle, we are of the view that variability is a *lifecycle aspect* and that aspect-oriented software development (AOSD) techniques can play an effective role in providing a holistic view of variability from its *conception* through to its *intention* and *realisation,* and its *effects* on *all* artefacts (e.g., code, plans, documentation, etc.) in a product line.

We propose combining traditional variability mechanisms, such as parameterisation or frame technologies, with AOSD techniques to modularise variable elements across the life cycle. Such an approach has proved successful in providing effective and flexible variability support at the implementation level [11]. However, it is also important that traceability of such parameters and aspects is maintained across the various abstraction levels, artefacts, development stages and granularities. We propose a variability framework whereby parameterised aspects, at various development stages and granularities, modularise generic concerns which are concretised by product-specific parameterisations. The parameterisation at various development stages and granularities are bound together in that one can propagate changes in a parameter at one level or granularity to parameters at other levels or granularities.

In our envisaged framework, *intra-dimension variabilities* in each of the dimensions above are captured by parameterised aspects. These aspects might exist at different levels of granularity, e.g., aspects capturing variability in each design model (fine granularity) and a set of aspects capturing variability across the design models (coarse granularity). A set of *inter-dimension aspects* maintain the traceability links that bind together the *intra-dimension aspects* and hence, the variability encapsulated by them. Again, such inter-dimension aspects may maintain traceability links at multiple granularities. Note that maintaining such traceability is a non-trivial task as our earlier studies have shown that aspects at higher levels of abstraction do not necessarily map onto aspects at implementation level [12]. For instance, *Availability* is an aspect at the requirements level which maps onto a decision for an architecture choice. Variabilities in availability requirements will lead to different architecture choices. Such architecture choices have to be further traced to design, implementation, test cases, etc. The problem is further compounded by the fact that a homogeneous set of techniques or artefacts is rarely used across the life cycle.

3. Conclusion

The inter-dimension and intra-dimension aspects collectively offer a framework which encapsulates the lifecycle aspect, variability. Of course, the design of such a framework is a challenging task. We have highlighted some key issues pertaining to homogeneity and traceability. In addition, there are other questions such as the role and nature of parameterisation at very high levels of abstractions such as requirements specifications or project management artefacts, e.g., documentation, reports, etc. Nevertheless, we are of the view that the development of such a holistic, multi-dimensional framework is worth exploring and that existence of such a framework would be of substantial benefit in the realisation of highly flexible software product lines.

References

1. Krueger, C., Variation Management for Software Production Lines, Proceedings of SPLC 2002 , pg 37-48.
2. van Gurp J. and Bosch J., Design Erosion: Problems & Causes, Journal of Systems & Software, vol 61, issue 2, 2002.
3. Redondo, P., et al, Supporting Software Variability by reusing Generic Incomplete Models at the Requirements Specification Stage, Proceedings of ICSR-8, 2004.

4. Kang, K. et al, Feature-Oriented Domain Analysis Feasibility Study, SEI Tech. Report CMU/SEI-90-TR-21, 1990.
5. Trigaux, J. and Heymans, P. Modelling Variability Requirements in Software Product Lines: A Comparative Study. Technical report PLENTY project, Institut d'Informatique FUNDP, Namur, Belgium, November 2003.
6. Gacek, C. and Anastasopoulos, M., Implementing Product Line Variabilities, ACM SIGSOFT Software Engineering Notes, v.26 n.3, p.109-117, May 2001
7. Fritsch, C., Lehn, A. and Strohm, T. Evaluating Variability Implementation Mechanisms, in Proceedings of International Workshop on Product Line Engineering, Seattle, USA, 2002.
8. Rouvello, I., et al, Business Users and Program Variability: Bridging the Gap, Proceedings of ICSR-8, 2004.
9. Concern Manipulation Environment homepage. http://www.research.ibm.com/cme
10. Lehman M. M., Ramil J. F. and Kahen, G. A Paradigm for the Behavioural Modelling of Software Processes using System Dynamics. Technical report Imperial College London, 2001.
11. Loughran, N., Rashid, A. Framed Aspects: Supporting Variability and Configurability for AOP, Proceedings of ICSR-8, 2004, pp. 127-140.
12. Rashid, A., Moreira, A., Araujo, J. Modularisation and Composition of Aspectual Requirements, Proceedings of AOSD 2003, pp. 11-20.