Designing Evolvable Software Products

Coordinating the Evolution of different Layers in Kernel Based Software Products

Project Description

Yvonne Dittrich and Peter Sestoft, June 2005

A software kernel is a part of software systems that is relatively stable over time and is used in a family of related software products. Software kernels are found in enterprise resource planning (ERP) systems, game engines, simulation software, content management systems, telecommunication infrastructure, and many other software products. A software kernel encapsulates and provides core functionality, and each specific software product is then developed by deploying and adapting this kernel.

The long-term goal of this project is to develop methods and technologies for the conscious identification, design and implementation of software kernels. This is a step towards rational development of families of software products, and should support an agile and competitive software industry.

Dissemination of the results and experience obtained in this project will happen through the proposed industrial collaboration, through a seminar series, through a summer school, through teaching at the IT University, and through publications.

The project focus is on development with software kernels, in particular these two aspects:

1. Identification of emerging software kernels. This work will be undertaken in collaboration with DHI Vand og Miljø (DHI), a Danish company that develops a range of software systems for simulation, all based on a well-established computational kernel for solving partial differential equations. However, the company now faces the challenge of identifying and developing a kernel for data organization, simulation setup, and graphical interaction with simulations and their results.

2. Technologies and methods that support both adaptability (to different application areas) and evolvability (over time) in a software kernel. This work will be undertaken in collaboration with Microsoft Business Solutions Danmark (MBS), a major developer of ERP systems. The company is a well-established supplier of such systems for small and medium-size companies, with a large number of partner companies that develop adaptations to the ERP kernel software for various sectors and trades. It is desirable to provide better support for evolution (upgrades) of the ERP kernel without breaking the adaptations made by hundreds of partners.

These two aspects will be investigated by one PhD student each, supervised by both senior researchers, who have complementary competences within software development methods and programming language technologies. Together with the involvement of industrial partners, this should ensure the methodological soundness, technological feasibility, novelty, and practical relevance of the proposed solutions.

Expected scientific contributions

We will explore and contribute to an approach to software development that has evolved in
practice but is only sporadically taken up in the research community. The proposed research has empirical aspects as well as technical and constructive aspects: what are the actual problems encountered in the participating organizations, and what **programming language technologies** and **software development methods** can help solve those problems in practice. We expect to further develop the research in these areas as well.

Many **programming language technologies** have been proposed over the last 40 years to support re-use in software development: function libraries, modules, interface specifications, object-oriented languages, single and multiple inheritance, component-based architectures, run-time reflection, and more. However, these technologies have turned out to either lack expressiveness or to poorly support static checking of the consistency of the resulting software. For example, run-time reflection provides for extreme flexibility but poor static checking, which means that many software errors are discovered only late, when they are most expensive to fix and most negatively impact the perceived quality of the software.

Programming language research has produced a range of concepts and tools that may help solve this problem. Among these are traits, mixins, type-based static checking (at compile-time, configuration-time or load-time), advanced module systems, aspect-oriented programming, model-driven development, and others. Some of these technologies have been known in academic communities for many years, but still appear to be little used in practice. We do have some initial ideas how traits, invented two decades ago [5] and now experiencing renewed academic interest [2, 12] can be used to address the adaptation and evolution problem. However, as far as we know, none of the published research on traits directly addresses the adaptation and evolution problem within and around software kernels.

From the perspective of **software development methods**, we note that any technical solution must match the needs for different dimensions of variability and change as well as the existing organisational setting and development practices. Existing methods for designing product line architectures focus on variation in functionality for the different members of a product line and on the technical separation between the fixed (reusable) and the variable (application-specific) parts of the application [22].

The development of software kernels and kernel-based applications often takes place in more than two tiers. For instance, sector-specific ERP standard systems are developed based on a kernel; these standard systems in turn are adapted to specific organisations; and finally, end-users can adjust settings to adapt their application to evolving requirements. Supporting and coordinating the continuous parallel evolution of different tiers of the software has not been addressed in research so far, nor has the coordination – technical and organisational – of more or less continuous parallel development of different tiers of a software product.

In an earlier project we observed this interlacing of different design activities between further development, tailoring and appropriation of applications for business areas that evolve rapidly [9, 10, 13]. These results challenge the traditional project-centered organisation of software engineering which does not sufficiently relate to the historical and organisational context of a specific project. Methods and processes for the development of kernel-based software must help coordinate the evolution of the different tiers. Changes have to be analysed and assigned to the right tier. The need for multi-dimensional variability has to be taken into account both when developing software products from scratch and when evolving an existing product.

**In summary**, we expect to show how a software development method that is based on the acknowledgement of evolutionary, multi-tier development, not only single projects, helps to
Designing Evolvable Software Products Appendix A – Project Description

separate and coordinate different layers of change and so promotes faster identification of the commonalities in a family of software products. Similarly, we expect to show how available programming language technology can be augmented with semantics-based tools to support the construction of evolvable software kernels.

The main challenge and the specific contribution of the proposed research is that it not only explores implementation technologies or development methodologies as separate entities, but also addresses how both are mutually conditional and together support the sustainable development of software products.

Related Work
As the proposed project addresses a real-world problem it is related to and builds on several research areas. We have therefore structured this section according to problem domains that we address in the project.

Developing and evolving kernel-based software products. Development and evolution of software based on kernels challenges the traditional software engineering focus on the single project. Though often organised as separate projects, the development within the different tiers is dependent on what is going on in the other tiers. Different versions of the kernel, different applications based on the same kernel, and the history of usage of the application, all provide input for the ongoing development.

Recently, the traditional mainstream of project-oriented software engineering, as promoted by [24], has been challenged by the agile development movement [3]. Development practices around small kernels might resemble agile development [16], but in the development of comprehensive kernels and applications, the ‘historical’ conceptualisation of software evolution [14, 20] seems more relevant. However, the research on software evolution does not address the issue of multi-tier development [1]. In earlier projects, we addressed the coordination between end-user tailoring and further evolution of applications in rapidly changing business areas [13] and observed development practices around a small software product [16], which provide a starting point for the research proposed here.

Design and implementation of software kernels. It is beneficial to develop a software kernel when part of the necessary functionality is stable and repeatedly used in several subsequent software versions and across a family of related software products or applications. That is, when there are identifiable commonalities and variabilities between the products, and the commonalities are stable over versions. We plan to compare the practice around the development of kernels with results from research on software re-use, software product lines, generative programming, and model-driven development [19]. Product-line architectures [26, 3] complement component-based software re-use by providing a common architecture for related software products. A concrete example is the model-based generation of software for families of embedded controllers, recently developed by one of our PhD students [29].

Metaprogramming, program generation [15] and generative programming [6] are relevant technologies for achieving the separation of the commonalities from the variabilities that is desirable in a software kernel. More generally, this separation between the constant and the variable is an example of the explicit consideration of binding-times, as known from partial evaluation [15] and from two-level languages [23]. The recent widespread industrial adoption
of optimizing portable execution platforms (Sun Microsystems's Java Virtual Machine and Microsoft's Common Language Runtime) has even made code generation at run-time feasible [27], although this technological opportunity seems to be underexploited in practice. We have considerable experience with reflection [18, 21], domain specific languages [17], virtual machines [7], generative programming [6], staging, binding-time separation, and partial evaluation [15]. There is need for better tools (type systems), for instance to ensure at compile-time the consistency of code generated at run-time. Research into this area is ongoing, nationally and internationally [28].

Program generation is far from the only way to implement a software kernel. The relation between the functionality provided by a kernel, the structure of the kernel, and the structure that it imposes on the applications based on it, is one important area where we expect to learn from product line approaches to software development.

**Identifying variability requirements and supporting adaptation.** Support for deployment and adaptation of software kernels is hardly researched at all. End-user development and end-user tailoring address the interlacing of different design activities with different actors – developer, user-close programmers, ‘super-users’ and end-users – during the lifetime of a software application [26]. We believe that the results of this discourse regarding tailoring interfaces, support, processes, documentation and co-operation for and around end-user development can serve as a good foundation for exploring the use and deployment of kernels, respectively the interfaces between different tiers and the requirements that these tasks pose for the design.

**Project Setup**

The cooperation with two companies that have very different software kernels allows us to compare different constellations of technical and methodological design. Comparing the two cases, we can begin to identify the factors that lead to the choice of different solutions both regarding technology and the organisation of the development processes. Future research can then be used to refine and extend the set of factors identified.

The two cases have very different business model, company size, kind of software, and end-user characteristics, so we decided to apply for two PhD projects, each one investigating the technical as well as the methodological aspects of one case. PhD scholarships posted at the ITU in general attract a large number of highly qualified applicants.

We expect both subprojects to implement the research approach outlined below. (See also time plan and Gantt chart in the description of the project organisation.) The specific focus of the empirical research will be decided together with the students and the companies, and might – depending on initial results – change during the project. The research approach and the project organisation is designed to handle the dynamics of such empirical and explorative research.

**Research Approach**

Our research approach takes as a starting point the ‘communities of practice’ [30] that evolved around the development and use of successful software kernels. This enables us to address the problems that had been encountered when rationalising the software development for a family of software products, and it allows us to generalise technical and methodological recommendations that are rooted in successful practices. To implement a cycle consisting of
empirical research and technical and methodological innovation, we apply a research approach called ‘co-operative method development’ [8], which has been applied in several projects [11]. Experience indicates that the approach provides a framework for co-ordinating empirical research with constructive innovations and for co-operation with partners outside academia. It can be understood as a domain-specific adaptation of action research, consisting of four phases where phases 2 and 3 can be applied repeatedly in an evolutionary cycle as outlined below.

**Phase 1:** The research starts with qualitative empirical investigations into the problem domain, in this case practices around development and use of software kernels as well as the design of existing kernels. The empirical research aims at understanding and explaining the practices and designs from a practitioner’s point of view, out of their historical and situational context, and to identify problematic aspects. In our case, we plan to investigate how the evolution of software products is organised today and what are the problems occurring. In parallel, we will analyse the technical design of the kernel-based products and the means to separate different dimensions of variability.

**Phase 2:** The results from the first phase are then used as an input for the identification of problematic aspects of the situation at hand and the design of possible improvements. This is done in co-operation between researchers and the practitioners involved. The researchers contribute with a survey of relevant research literature. On the one hand we will investigate what literature tells about processes of software evolution, reuse and product line development. On the other hand we will investigate programming language technologies, such as generative programming, traits, static checking and other semantics-based tool support, for the implementation and management of variability. The literature studies will be complemented with exploratory prototypes giving a better understanding of different techniques. The result of this phase is the design of measures that can be expected to improve the situation at hand and address some of the problems identified.

**Phase 3:** The improvements will be implemented. Methodological and technical improvements developed in phase 2 may be implemented in smaller projects at the industrial partners. The researchers will accompany these method improvements as participatory observers. In case where technical improvements cannot be implemented in the actual software products, we plan to implement a set of ‘proof of concept prototypes’ implementing an alternative kernel using different technologies.

**Phase 4:** Finally the results are evaluated together with the practitioners involved. The result of this evaluation will both summarize concrete results for the companies involved and build the base for the scientific evaluation of the proposed improvement measures for the research project.

**References**


