Software Evolution in Practice: Adding Web Functionality to a Legacy System

Michalis Anastasopoulos[1], Joachim Bayer[1], Christian Bunse[1], Jean-François Girard[1], Isabel John[1], Dirk Muthig[1], Peter Sody[2] and Enno Tolzmann[2]

[1]Fraunhofer Institute Experimental Software Engineering (IESE) Sauerwiesen 6, D-67661 67661 Kaiserslautern, Germany
{anastaso, bunse, bayer, girard, john, muthig}@iese.fhg.de

[2] Sieda GmbH Richard Wagner. Str, D-67655 Kaiserslautern, Germany
{sody, tolzmann}@gwi-ag.com

Abstract. Software evolution requires an intelligent planning of evolution and maintenance activities in an organization. Intelligent planning means integrating the long-term strategy of a product's evolution with short-term maintenance activities. This paper describes a successful example of software evolution in the context of the applied-research project APPLICATION2WEB. The paper describes experience in transferring technology into a small company to realize web services based on a legacy system. The project covered all necessary activities including reverse engineering, domain analysis, architecture analysis, and middleware integration.

1 Introduction

Software evolution immediately starts when a software product has been shipped the first time. Software evolution, thereby, generally is the continuation of the product's development. That is, software evolution subsumes a range of activities from realizing new or changed requirements down to fixing small bugs in the code. The key difference between the development of a product from scratch and the evolution of a product is that a developer must ensure compatibility with former releases of the same product. That is, the developer must fulfill the users' expectation that the system improves but everything that worked well before should still work in the same way the users are used to it.

These constraints require an intelligent planning of evolution and maintenance activities in an organization. Intelligent planning means integrating the long-term strategy of a product's evolution with short-term maintenance activities. If the long-term strategy is explicitly available, it enables the development team to streamline all evolution activities and thus keep product evolution as efficient as possible. Although simple in theory, defining a long-term strategy is an extremely difficult task because requirements and trends of the future must be anticipated in advance.

In this paper we describe experience with technology transfer into a small company which led to the realization of web services based on a legacy system and to a partial

redesign of components of the legacy system including activities in reverse engineering, domain analysis, architecture analysis and middleware integration.
The remainder of this paper is structured as follows: in section 2, the project context where the technology transfer was made is introduced. In the following sections, the activities in modeling, reverse engineering, architecture and middleware are described. The results of these activities and experiences and lessons learned are described in Section 7, Section 8 concludes the paper.

2 Project Context

In this section we will introduce three things: first, the research project APPLICATION2WEB that defined the context in which the technology transfer happened, second, the company Sieda, and third, the goals of the cooperation.

2.1 APPLICATION2WEB

APPLICATION2WEB is a project funded by the German ministry of economics. The goal of the project is it, to give small and medium sized companies the possibility to offer and sell their products and services on the web in an efficient way and at reasonable costs. In the project existing technologies and techniques should be exploited and extended to make it possible to companies to launch new applications with web functionality based on their legacy products. So the goal of the project is to support companies in the evolution of their old products and the realization of new planned web functionality
The consortium of the project consists of two research institutes (Fraunhofer IESE, Kaiserslautern and Forschungszentrum Informatik FZI, Karlsruhe) and 8 small and medium companies (including SIEDA) as application partners. The participating research partners (IESE, FZI) supported the respective application partners in applying the technologies that were useful in the respective context. This was done by:

- Enabling the selection of the right technology with decision support in the company specific situation
- Supporting the tailoring of technologies to the environment and the specific goals and problems of the company
- Consulting during the application of the new technologies and methods
- Supporting the packaging of the results

These technology transfer steps were also performed by Fraunhofer IESE and Sieda and led to the results described in this paper.

2.2 Sieda

The company SIEDA – Systemhaus für intelligente EDV Anwendungen GmbH was created 1993 in Kaiserslautern and has a staff of 14 employees at the moment.

The SIEDA GmbH is a specialist for the development of intelligent software for the solution of complex optimization problems in particular short-time manpower planning. Their main system “Orbis Dienstplan” (short Orbis) is a roster planning system with application emphasis in the public service. Contrary to most products available at the market it is specialized in planning by multi-layer models (e.g. with services around the clock). For intelligent planning constraint-solving techniques are used, as described in [1]. Apart from the actual planning also the performed services are entered through the system and the data for the wage and salary statement can be computed with consideration of tariff regulation. Fig. 1 depicts the overall process supported by the software. It consists of three major activities: planning of the roster, time registration, and analysis of the registered times with respect to the roster in order to generate input to an external payroll system.

2.3 Goals

The overall goal of the activities of Sieda within APPLICATION2WEB was to enable internet-based access to the roster software system. To do so the right components had to be identified to integrate the web functionality efficiently and cleanly. Thereby, the overall architecture of the system should stay unchanged as far as possible.

Hence component-based technologies (with an emphasis on reuse and product line engineering) were in the focus of the collaboration between IESE and Sieda. The goals in more detail were:

- *Selection and redesign of central components*: Central component of the roster software system should be selected, which allow an easy linking of a web interface to the existing system.
- *Integration of a web interface:* A web interface should be planned, designed and integrated as supplementary and variable component.
- *Legacy Integration:* Web interface and legacy system should be integrated. The functionality of the legacy system should be supported by the web system
- *Reduction of the system’s complexity*: Through redesign of a central component, redefinition of the interfaces, and a reduction of coupling and cohesion, the overall complexity of the system should be reduced significantly.
- *Improvement of the system’s usability*: Parallel to the web interface, parts of the existing user-interface should be redesigned.
- *Improvement of the system’s maintainability*: All the above goals should contribute to a decrease in overall maintainability.

According to the web interface or web application there were two scenarios considered, personalized access and application service providing. With personalized access every employee should be enabled to individually interact with the system. The set of useful services ranges from scenarios mainly providing access to individual data (e.g. making monthly payments especially bonuses more comprehensible) to complex interaction scenarios, such as exchanging working times and locations. In the ASP scenario it should be possible to offer satellite stations access to the latest version of the roster and make an autonomous plan for the satellite stations.

In the following sections we will describe the activities that were performed within the project. There were activities in the area of domain analysis/requirements modeling, reverse engineering, architecture stratification and integration of middleware. All activities have a different focus but the same goal, the reduction of the complexity of the system with a concurrent integration of web functionality.

3 Domain Analysis

The goal of any domain-analysis approach is to identify and document requirements on a set of systems in the same application domain. In order to successfully introduce such an approach, the target organization must already master requirements engineering for single-systems according to our past experience [2].

If this is the case, in an organization, there is typically either one requirements document capturing the requirements on the "average" system, or a suite of requirements documents each capturing the requirements on different single systems. Domain analysis methods provide processes for eliciting and structuring the requirements of a domain, or product line. This can e.g. be done with a product line engineering approach like PuLSE^{TM}[3]. The results are captured in a domain model. A domain model must capture both the common characteristics of the products and their variations.

The domain analysis-process we followed here consists of two steps, sub-domain identification and analysis of the selected sub-domain. The sub-domain identification step of our domain-analysis approach analyzes all existing request that determine the next maintenance and evolution activities in an organization. That is, initially input from maintenance, marketing, and application engineering projects is collected. On this basis, a prioritized list of good sub-domain candidates is compiled. In the analysis step, the sub-domains selected as subject of the domain analysis are then mapped to logical (potentially generic) components of the software. Thereby, a logical component is somewhere between the theoretical, ideal component that exactly supports the sub-domain functionality and the set of existing physical software components, that actually realize the sub-domain's features in practice. For each selected logical component, the recursive modeling and refinement process of the KobrA method is ap-

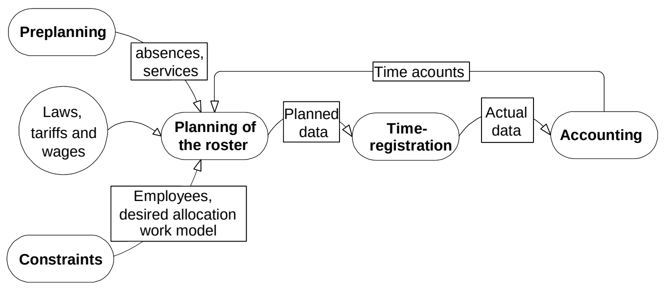


Fig. 1 Software-supported business process of the Orbis system

plied [4]. This process concurrently models two aspects: on the one hand, the data relevant for or manipulated by logical components and, on the other hand, the activities in which the considered logical components are involved. The integration of the two aspects results in an object-oriented domain model that documents a domain's (generic) data, behavior, and functionality
From the marketing perspective, the need for providing system services via the Internet was the main requirement for the next system generation (c.f. Section 2.3). The application domain defined by the process described above is of an average size and thus is too large to be analyzed as a whole in a single domain analysis step. Everyone who has ever studied German wage agreements can easily confirm this statement. Therefore, we selected accounting, the last of the three major process steps (c.f. Fig. 1) of the Orbis system as subject of our domain analysis effort. In several recursions of the DNA process, further sub-domains of the accounting domain were identified and analyzed.
In our context, we had to include web usage scenarios. Due to the latter realization, we decided to first analyze the accounting manager interface from the existing report writer's point-of-view and later build a client on top of this initial interface definition.

4 Reverse Engineering

Web-enabling a client-server application efficiently requires a clear vision of the existing application and its future. In this project two ways were used to obtain this vision: reverse engineering the Orbis system and analyzing different architecture alternatives for the future.

Reverse Engineering Orbis

The goal of the reverse engineering activities was to map the relations and dependencies present in the code onto the current conceptual architecture description and onto the alternative future system structure. The mapping to the current description assures that the transformation of the system does not start from an idealized abstraction of the system. The mapping to potential future system structure offers a first approximation of the effect of the new structure on the existing dependencies among the components, before the classes are re-factored or adapted to the new structure.

Fig. 2 depicts the way to produce these two mappings. The first step was to extract the relations and dependencies (e.g. method calls, data usage) from the source code and to represent them in the flexible relational *rsf* format of the rigi tool [5]. The next step was to capture the current conceptual architecture, map classes and directory to the component of this architecture. The final step was to visualize and analyze the actual relations and dependencies of the code on this architecture description. The visualization with rigi and applying a customized version of the reflexion model approach [6] allowed unexpected connections among components to be discovered incrementally

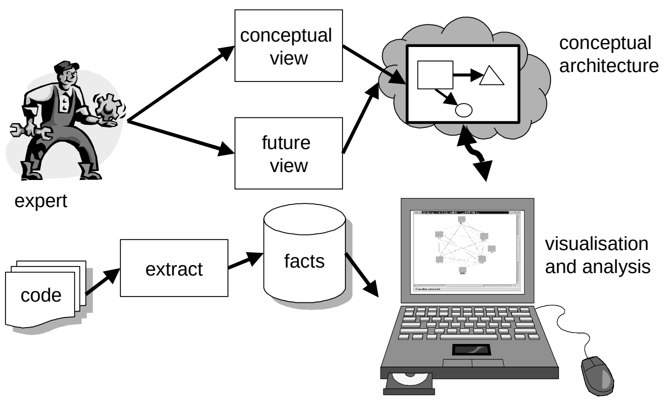


Fig. 2 The mapping process during reverse engineering

and interactively, as well as the relationships behind these connections. The mapping of the future system structure proceeded in a similar way.

Analyzing early Architecture Alternatives

Choosing among potential architecture alternative to achieve business goals is often a difficult task, because of the number of interacting aspects to consider. To support this decision, we performed a customized version of architecture trade-off analysis method (ATAM [7]). The goal of this architecture analysis is to verify a software architecture against its requirements (functional and non-functional), that is, architecture analysis aims at determining if, to what degree a software architecture satisfies its most important requirements and to identify risk and associated with the various architecture alternatives.

In the concrete project context, we proceeded as follows:

- First the architect produced a sketch of 4 different architecture alternatives.
- Then we elicited the envisioned web versions of Orbis (personalized access/ASP c.f. Section 2.2), the various stakeholders together with their perspective on the system, and a quality tree relevant to these applications.
- Based on this quality tree and on the stakeholders' perspective we generated the 25 functional and 33 non-functional requirements. The functional requirements were captured through use cases. The non-functional requirements were formalized as different scenarios (e.g., usage, modification, usability, deployment, and performance).
- The scenarios and the use cases were then prioritized and the highest priority scenarios were applied to the various architecture alternatives. As a results a list of issues and risks were produced and the most appropriate alternatives were identified

The architect identified architecture approaches, styles and patterns based on the scenarios and use cases gathered. These form a useful input for refining the architecture sketch into designs.

5 Architecture Stratification

The problem in describing the interaction between two components is that this simple idea can span a wide range of abstraction levels. At one end of the spectrum there are very high-level interactions between the user and the system, perhaps involving a significant proportion of the system's processing elements (e.g., use cases). At the other end, there are most detailed interactions between operating-system objects. Some stakeholders of the system usually want to think in terms of high-level abstractions (e.g., architects, customers, managers), while others (e.g., implementers, testers, maintainers) need to deal with the lowest level that provides full details. Directly defining the system at the most detailed level is therefore insufficient because it fails to provide one set of stakeholders with an appropriate view of how the system works.

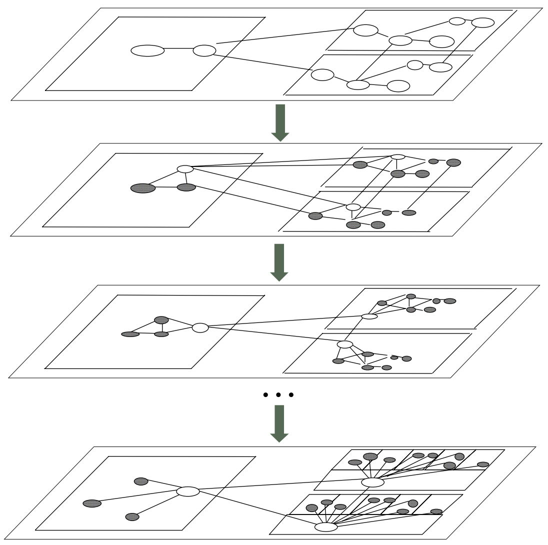


Fig. 3 Stratified Architecture Structure

The principle of architecture stratification [8] provides a flexible and component-oriented way of separating the description of distinct aspects of a system. The basic idea is to represent a community of components at several levels of architectural abstraction, *each* providing a *complete description* of the system, and each related to the layer above by explicit interaction refinement. The top level in this hierarchy represents the most abstract view with the most generic description of the interactions between component instances. The bottom level represents the most detailed view, with the most detailed representation of the interaction relationships.

In general, a complete stratified architecture is comprised of multiple strata as illustrated in [8]. Each stratum represents a refinement of the stratum above, and thus typically contains additional components and interactions that explain how interactions in the higher strata are realized. The abstraction levels in a stratified architecture range from a high level, analysis-like description of the system down to the most detailed implementation code. The highest level will be most useful for understanding the overall solution strategy. In the lowest stratum one will find the usual complex web of objects where it is hard to "see the forest for the trees". However, this is often the only appropriate level of abstraction to resolve realization details. Depending on the application domain, the same sequence of types of strata will be useful to structure a system's architecture according to its emergent (often non-functional) properties.

Overall, stratification is important in every application domain, but is particularly critical for web-based applications that are inherently distributed, and thus embody the multi-protocol stacks and communication realization layers that are characteristic of distributed systems. Architecture stratification helps to separate the layers without introducing restrictions as to what lower level functionality is available to high-level components. Web-based applications also tend to be more heterogeneous than applications in most domains, and thus have a greater need for distinct levels of abstraction and interface wrapping concepts. Stratification offers a clean model for capturing different aspects of a system's implementation in a way that is not only compatible with component technology, but also lends itself to realization in the form of generic component frameworks. The ultimate benefit of the approach is therefore to enhance opportunities for maintenance and reuse.

As a first step in architecture stratification, a component (preplanning, German: Vorplanung, c.f. Fig. 1) of the existing system was selected and analyzed in a case study. The preplanning component provides users with a calendar with their currently planned roster in which they can enter wishes for changes (e.g., vacation, certain shifts on certain dates). These wishes are then later taken into account when the actual roster is planned. The goal of the case study is to prepare this component for being accessible via the web. A read-only version of preplanning exists already (i.e., users can access their personal roster via the web). In the case study this version is to be extended for writing user access.

The strata that were selected and modeled using the UML were:

- Context: In this stratum the preplanning component context (i.e., the components it interacts with) is modeled. This stratum was mainly chosen to completely understand and explicitly model the requirements on the preplanning component.
- Top Layer: This stratum contains the description of the preplanning component (i.e., identifies subcomponents and their interactions).
- Persistence: This stratum identifies the components and classes of the top layer that are being made persistent and describes the mechanisms for doing that.
- Communication: In this stratum, two alternatives for distribution preplanning are explicitly modeled (i.e., via http protocol or using Active Server Pages).

6 Middleware

The main requirement of the different planned web applications (c.f. Section 2.2) was the smooth integration of legacy business logic, which is available mainly in the form of native dynamic link libraries. Furthermore, the web application had to enable the management of user accesses and sessions. The former includes the definition of user accounts and groups as well as user authentication and authorization.

Given these functional requirements it became necessary to set up a three-tiered architecture as illustrated in Fig. 4. As shown above the middle tier assumed the responsibilities of delivering and generating HTML as well as managing users and their sessions. Moreover the new middle tier should fulfill a set of non-functional requirements including scalability and ease of configuration and deployment.

Given the company-specific situation the effort and especially the financial cost for setting up the middle tier was to be kept as small as possible. This has quickly led the architects towards open source solutions that have gained popularity lately because among other things they can be obtained free-of-charge and they provide a series of ready-to-use services (including user and session management). After evaluating a

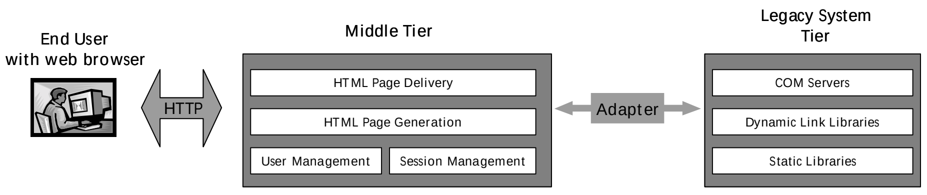


Fig. 4 Architectural tiers

number of open source products it had to be decided between Apache and Zope. The latter appeared less appealing because it is based on Python, a programming language not familiar to the developers and because it uses proprietary technologies thereby not being compatible with many open standards.

The Apache web server, which according to latest statistical information [9] is the most popular server, was therefore preferred. Apache has features for high-performance, scalable HTML delivery and it provides a simple mechanism for managing users.

One of the reasons for the big success of the Apache server is also the variety of Apache-based frameworks that cover a broad spectrum of different requirements for web applications. Tomcat is such a project, which was selected for generating dynamic web pages and for managing user sessions through the use of the Java language.

The main requirement however, which, as mentioned before, was the legacy integration, had to be implemented from scratch. Since the dynamic HTML content is generated by Servlets, which are Java applications, it was decided to use the low-level JNI (Java Native Interface) API. To this end it was required to build the Adapter shown in Fig. 4, which is a native application providing the JNI-compliant view of the legacy libraries.

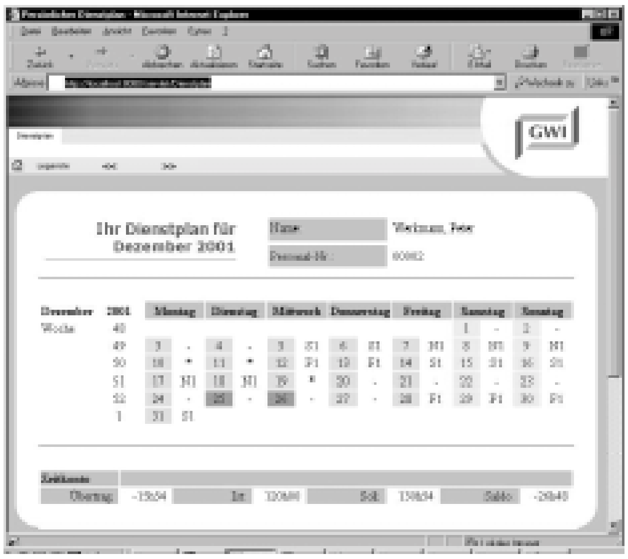
7 Results

7.1 Overall results

The main goal, to allow Internet based access to the roster software system of SIEDA was achieved. A web client (called "Mein Dienstpl@n") was implemented, based on the results of the domain analysis and supported by the architecture and reengineering efforts in APPLICATION2WEB. A snapshot of the web client can be seen in Fig. 5. With this web client a nurse can check via the intranet or internet which services she had in the last weeks, what services are planned for the near future and how the services she provided are accounted. The easy and fast integration of the new component showed the flexibility of the system. This "read-only"-Web client is only a first step towards more complex internet-based services for the roster software.

The reduction of the overall system complexity through the described methods could not be proven by objective measures. This is one of our major goals of future activities within the project context. Maintainability was primarily improved through the reverse engineering an stratification efforts. Discovering the relationships behind the system's components and documenting them could reduce maintenance effort. As far as the accounting domain and related data was concerned, the existing user interface was redesigned. The concepts that were clarified during our analysis were also realized in the user-interface of the existing application.
Fig. 6 shows the architectural evolution throughout the whole project that was supported by all our activities.

To summarize, most of our goals were attained (selection and redesign of a central



Ihr Dienstplan für Dezember 2001

Name	Weixman, Peter
Personal-Nr.:	60082

Dezember	2001	Montag		Dienstag		Mittwoch		Donnerstag		Freitag		Samstag		Sonntag	
Woche	48											1	-	2	-
	49	3	-	4	-	5	S1	6	S1	7	N1	8	N1	9	N1
	50	10	*	11	*	12	F1	13	F1	14	S1	15	S1	16	S1
	51	17	N1	18	N1	19	*	20	-	21	-	22	-	23	-
	52	24	-	25	-	26	-	27	-	28	F1	29	F1	30	F1
	1	31	S1												

Zeitkonto							
Übertrag	-15h54	Ist	320h00	Soll	150h54	Saldo	-29h40

Fig. 5 A snapshot of the web client

component, integration of a web interface, improvement of overall usability and maintainability) and evolution of the system was supported.

7.2 Lessons learned

Besides this project specific results and the achievement of the project goals there are also more general observations on the benefits and problems of technology transfer approaches in the areas of domain engineering, reverse engineering, architecture and middleware.

For all parties involved it showed out to be useful and helpful to do domain analysis for web-related purposes. Although the requirements on the web-interface were rather unclear at the beginning, the domain analysis we made helped in a very straightforward manner to clarify and realize the web interface.

Another observation we made, which surely is not new and was made in many contexts already was that the people from IESE need a minimal amount of domain knowledge to start with the selection and application of their approaches and domain knowledge is not so easy to achieve.

It was not easy to find the right domain to focus on for a domain analysis. Other domains represented in the roster system, like preplanning, time registration or reporting we could also have chosen as domains for the domain analysis. But by carefully and not ad-hoc selecting the accounting domain we could optimally prepare for the web-interface and reduce coupling and cohesion of the overall system. So, taking some time in selecting the right part of a system for a domain analysis, as generally there is not enough time to do a thorough analysis really pays.

It is important to tailor/customize the approach that should be transferred to industry. Practitioners should be able to learn new technologies based on the things they already know. So an incremental approach, starting with well-known technologies and integrating more and more new methods (like techniques for handling variability) is the right approach.

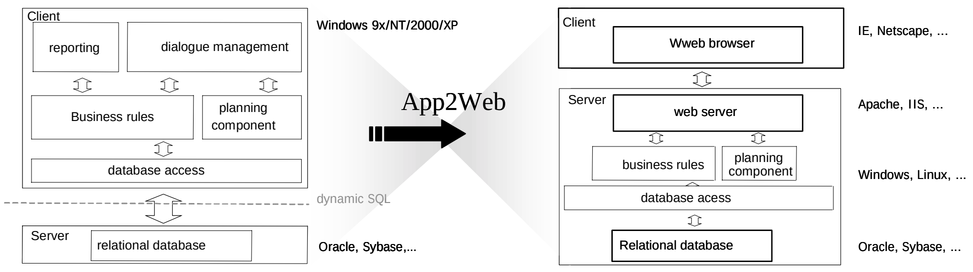


Fig. 6 Architectural evolution

8 Conclusions

In this paper we showed how technology transfer was performed with the purpose of the integration of web-functionality into an existing system. We described the surroundings of the project, how we combined and applied several techniques and approaches to achieve the goals of the collaboration, and summarized the main results of the technology transfer.

The goals of the project have been achieved in a straightforward and efficient way. Unfortunately, exact numbers and measures to document our success in an objective way was not possible due to the hard time- and budget-constraints. Hence, it was not possible to set up a measuring program to empirically prove our statements and experiences. So when continuing our project and for further observations it is desirable to set up a small measuring program to collect some data and empirically validate our goals and thus enable other companies to repeat the results we retrieved in their context systematically and successfully.

References

1. Harald Meyer auf'm Hofe: Solving Rostering Tasks as Constraint Optimization; in: Burke, Erben (eds.), Selected Papers from PATAT-2000 , Lecture Notes on Computer Science (LNCS), Springer Verlag, 2000.
2. P. Knauber, D. Muthig, K. Schmid, and T. Widen. Applying Product Line Concepts in Small and Medium-Sized Companies, in IEEE Software, Sep./Oct. 2000
3. J. Bayer et. al. PuLSE: A Methodology to Develop Software Product Lines, in Proceedings of the Fifth Symposium on Software Reusability, ACM, May 1999
4. C. Atkinson et. al. Component-based Product Line Engineering with UML, component series, Addison-Wesley, 2001
5. H. Müller, K. Wong, and S. Tilley. Understanding software systems using reverse engineering technology. In 62nd Congress of ACFAS, Montreal, May 1994
6. Gail C. Murphy, David Notkin, and Kevin Sullivan. Software Reflexion Models: Bridging the Gap Between Source and High-Level Models In *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, October 1995, ACM, New York, NY, p. 18-28.
7. R. Kazman, M. Klein, P. Clements, „ATAM: Method for Architecture Evaluation“, CMU/SEI-2000-TR-004, SEI, October 1999
8. Colin Atkinson and Thomas Kühne Aspect-Oriented Development with Stratified Frameworks, Accepted for "IEEE Software", 2002.
9. Netcraft Web Server Survey, Market Share for Top Servers Across All Domains August 1995 - May 2002, http://www.netcraft.com/survey/