Integration of Software Systems – Process Challenges

Rikard Land, Ivica Crnkovic, Christina Wallin

Mälardalen University
Department of Computer Science and Engineering
PO Box 883, SE-721 23 Västerås, Sweden
+46 21 10 70 35, +46 21 10 31 83

{rikard.land, ivica.crnkovic, christina.wallin}@mdh.se
http://www.idt.mdh.se/{~rld, ~icc}

Abstract

The assumptions, requirements, and goals of integrating existing software systems are different compared to other software activities such as maintenance and development, implying that the integration processes should be different. But where there are similarities, proven processes should be used.

In this paper, we analyze the process used by a recently merged company, with the goal of deciding on an integration approach for three systems. We point out observations that illustrate key elements of such a process, as well as challenges for the future.

Keywords

Software Architecture, Software Evolution, Software Integration, Software Process Improvement.

1. Introduction

Software integration as a special type of software evolution has become more and more important in recent years [7], but brings new challenges and complexities. There are many reasons for software integration; in many cases software integration is a result of company mergers. In this paper we describe such a case, which illustrates the challenges of the decision process involved in deciding the basic principles of the integration on the architectural level.

2. Case Study

Our case study concerns a large North-American industrial enterprise with thousands of employees that acquired a smaller (~800 employees) European company in the same, non-software, business area where software, mainly in-house developed, is used for simulations and management of simulation data, i.e. as tools for development and production of other products. The expected benefits of an integration were increased value for users (more functionality and all related data collected in the same system) as well as more efficient use of software development and maintenance resources. The first task was to make a decision on an architecture to choose for the integrated system. The present paper describes this decision process.

Figure 1 describes the architectures of the three existing systems in a high-level diagram blending an execution view with a code view [3]. The most modern system is built with a three-tier architecture in Java 2 Enterprise Edition (J2EE), while the two older systems are designed to run in a Unix environment with only a thin "X" client displaying the user interface (the "thin" client is denoted by a rectangle with zero height in the figure); they are written mostly in Tcl and C++, and C++ with the use of Motif. The Tcl/C++ system contains ~350 KLOC (thousands of lines of code), the C++/Motif system 140 KLOC, and the Java system 90 KLOC. The size of the rectangles in the figure indicates the relative sizes between the components of the systems (as measured in lines of code). The Tcl/C++ system uses a proprietary object-oriented database, implemented as files accessed through library functions, while the two other systems, which were developed at the same site, share data in a common commercial relational database executing as a database server.

Since the two software development departments (the North American and the European) had cooperated only to a small extent beforehand, the natural starting point was simply to meet and discuss solutions. The managers of the software development departments accompanied by a few software developers met for about a week, outlined several high-level alternatives and discussed their implications both in terms of the integrated system's technical features and the impact on the organization. Since the requirements for the integrated system was basically to provide the same functionality as the existing systems, with the additional benefits of having access to more and consistent data, user involvement at this early stage was considered superfluous. At this meeting, no formal decision was made, but the participants were optimistic afterwards – they had "almost" agreed. To reach an agreement, the same managers accompanied with software developers met again after two months and discussed the same alternatives (with only small variations) and, once again, "almost agreed". The

same procedure was repeated a third time with the same result: the same alternatives were discussed, and no decision on an integrated architecture was made. By now, almost half a year had passed without arriving at a decision.

Higher management insisted on the integration and approved of a more ambitious project with the goal to arrive at a decision. Compared to the previous sets of meetings, it should contain more people and involve more effort, and be divided into three phases: "Evaluation", "Design", and "Decision", with different stakeholders participating in each; see Figure 2. First, the users were supposed to evaluate the existing systems from a functional point of view, and software developers from a technical point of view. Then, this information should be fed into the second phase, where software developers (basically the same as in phase one) should design a few alternatives of the architecture of an integrated system, analyze these, and recommended one. In the last phase, the managers concerned were to decide which architecture to use in the future (maybe, but not necessarily, the one recommended in phase 2). The first phase lasted for two weeks, while the second and third phases lasted for one week each.

Of course, this characterization is somewhat idealized – in reality, there were more informal interactions between the stakeholder groups and between the phases: briefings were held almost each day during the course of the meetings, to monitor progress, adjust the working groups' focus etc.

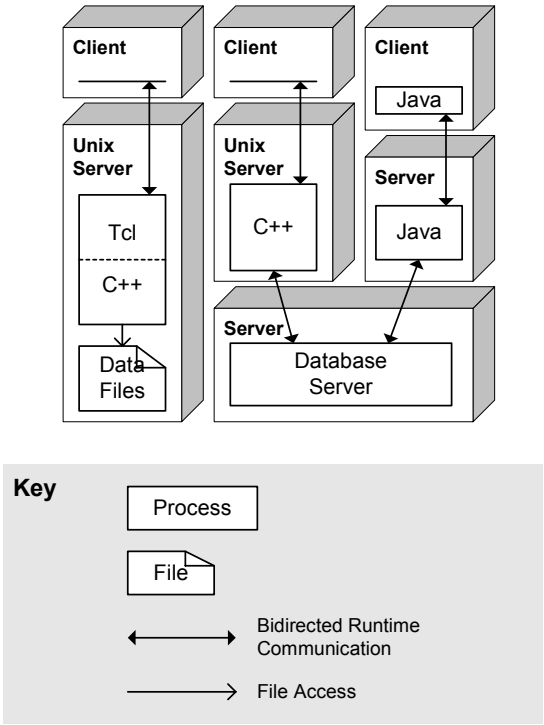

Figure 1. Today's three systems.

Phase 1: Evaluation. Six users experienced with either of the three systems had hands-on tutorials and explored all the existing systems, guided by an expert user. They produced a high-level requirements specification with references to what was good and less good in the existing systems. In general they were content with the existing systems and were explicit in that it was not necessary to make the user interface more homogeneous; they would be able to work in the three existing user interfaces, although very dissimilar. The user evaluation would therefore not affect the choice of architecture.

The developers found that although the existing systems' documentation included overall system descriptions, they were of an informal and intuitive kind (for example, none of them used UML), which meant that the descriptions were not readily comparable, making the development of architectural alternatives difficult. During the first phase, the developers were therefore to produce high-level descriptions of the existing systems that would be easily comparable and "merge-able".

Phase 2: Design. In phase 2, the software developers tried several ways of "merging" these architectural descriptions. Their experience and knowledge of the existing systems was the most important asset. Two main alternatives were developed, a "data level" integration (preserving the differences between today's systems but adapting them to use the same database, see Figure 3a), and the "code level" integration alternative (using the three-tiered architecture of the existing Java system, see Figure 3b). The architectural descriptions were analyzed briefly regarding functionality and extra-functional properties such as performance, maintainability, and portability, and project plans for the implementation of the two alternatives were outlined. The developers recommended the "code level" alternative due to its many perceived advantages: it would be simpler to maintain, bring the users more value, be perceived by users as a homogeneous system, while not being more expensive in terms of effort to implement (according to the estimations, that is).

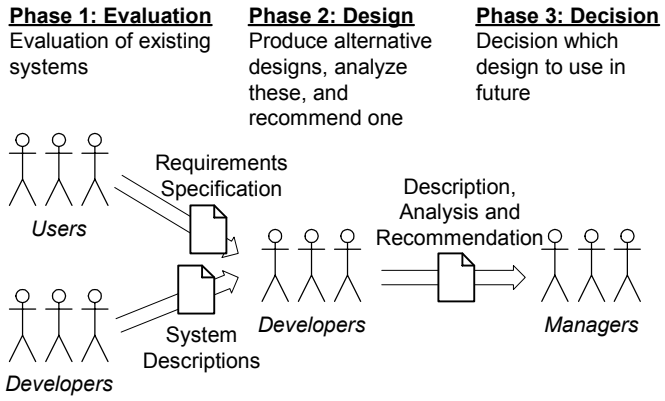

Figure 2. Project phases.

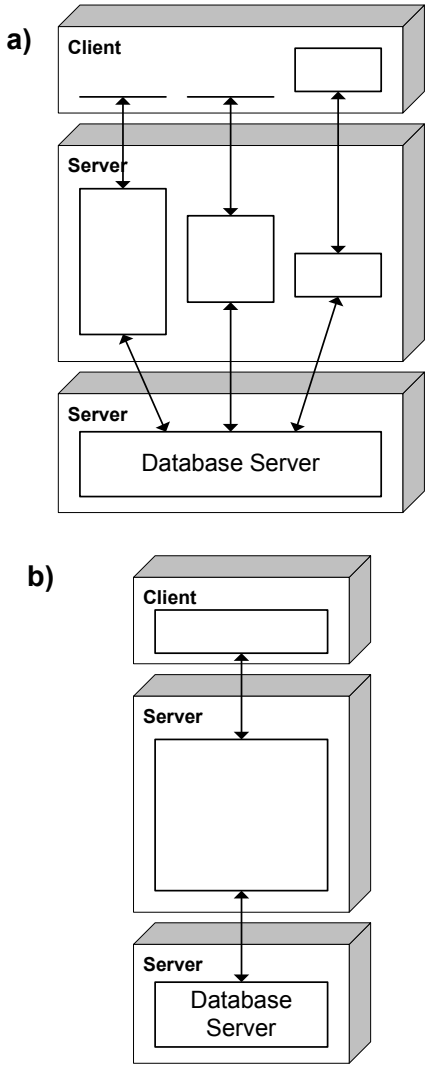

Figure 3. The two main integration alternatives.

Phase 3: Decision. All written documentation (architectural descriptions, project plans for their implementation, and other analyses) was forwarded to the third phase. The managers concerned had a meeting for about a week when they discussed costs, risks, business implications, organizational impact, etc. of the two alternatives. It was decided that the systems should be integrated according to the "data level" alternative, since this solution was considered to be associated with a lower risk than the "code level" alternative; risk meaning the probability of overrunning budget and/or schedule, producing a product of poor quality, or fail altogether with the integration. The risk parameters are not only those related to technical problems (such as those involved with writing new code), but also the risk of successful collaboration (in terms of "commitment required" from departments of two previously separate organizations, not yet so close collaborators).

3. Analysis

While a handful of alternatives were discussed during the first meetings, there were only two alternatives produced in the design phase of the three-phase project. The alternatives themselves were not new – the developers almost indignantly said that they discussed the same alternatives and issues as they had done for six months. It was rather the ability to agree on discarding some alternatives with a certain amount of confidence that was an improvement as compared to the first sets of meetings. Assuming that the developers were correct in that the discarded alternatives were inferior, this reduction of the numbers of alternatives was arguably an improvement compared to the first sets of meetings. The managers in the third phase had "only" to choose between these two alternatives, and as we described, the users did not favor any of these, which made it possible for the managers to base the decision on a smaller set of concerns.

In the rest of this section, the features of the process that enabled these improvements are discussed. We highlight what we believe to be good practices in general during software integration as well as challenges for the future. These conclusions are partly based on a questionnaire responded to by (some of) the participants of the projects.

Early meetings. In a newly merged organization, the "people aspect" of software integration needs to be addressed, and meeting in person to discuss integration in general, and even particular alternatives, is the most important means to build the trust and confidence needed. This should not be seen as a replacement for a more structured project, however.

Several-phase process. By dividing the stakeholders into different activities with specific tasks, the discussions become more focused and efficient. At the same time, more interaction that only forwarding deliverables is needed; in the project, briefings were held almost every day involving people concerned, to monitor progress and adjust focus if needed. The scheme used does not differ from already documented good practices in other software activities, such as development and maintenance.

User involvement. Performing a user evaluation of existing systems prior to integration is crucial. If the outcome does not affect the choice of architecture, this is good news for the decision process – the choice can be made based on other concerns. Moreover, any issues found during the user evaluation are important inputs to subsequent phases, during actual implementation. Since the user evaluation did not affect the choice in the case study however, it did not really fulfill the developers' expectations. We therefore suggest that in an integration process the expectations should be clearly articulated. If the goal of the user involvement at this early stage is to assess whether they have any preferences that affects the choice

of architecture, the type of evaluation performed in the case study seems reasonable – enough users must be given time to understand the systems in enough depth to achieve a certain amount of confidence in the analysis results. However, if the goal is to take the opportunity of improving the existing systems significantly when integrating them, the situation reminds of development of new software, and established requirements engineering, more heavily involving users and other stakeholders, should then be applied [4]. The existing systems can be thought of as a requirement specification or prototype in evolutionary or spiral development [1]. A cheap, initial investigation involving users may indicate that a more thorough evaluation is needed.

Separating Stakeholders. This should be no surprise – it does not make sense to bring all stakeholders together for all meetings during the process. We have showed a three-phase process where the separation of stakeholders made the meetings more efficient and focused. The discussions were kept at a level detailed and technical enough to enable fruitful discussions since the participants had similar background and roles. By assigning different tasks to the different phases, the responsibilities became clearer. The developers could first concentrate on evaluating the existing systems, and only later bother about their integration. The managers were reduced to "only" making a decision, basically by choosing between two alternatives with certain properties.

Active upper management. Upper management insisted that the systems should be integrated: implicitly, since they once again started a project with the same goal, and more explicitly by deciding on a date when there had to be a decision. There was an integration coordinator, responsible for all integration activities resulting from the company merger, who actively showed interest in the project.

Architecture-centric process. During many software activities, the process can benefit from being oriented around the architecture of the system being built [8]. How the architecture was used in this particular case study has been described in more detail elsewhere [5,6].

Different people. Although there were developers and managers participating in each project execution the people participating in each meeting or in the final project were not identical. Perhaps the mix of people in the successful project was a successful blend of open minds, while in the previous meetings this was not the case? According to the questionnaire data, this might be the case.

It will take time. Eight months passed from the initial meetings to the decision. This means that the project members and the managers had got to know each other better on a personal level, and overcome cultural differences between the two countries and formerly separate organizations [2]. When a decision is dependent on people collaborating for the first time, especially when they have different cultural backgrounds (as is the case after mergers, especially international ones), it must be expected that the process will take more time than a project executed completely within either of the departments – and possibly also a higher amount of disagreement and frustration. With this in mind, it is likely that the actual integration also will take time, and that an integration project in the context of a company merger will face more obstacles in terms of cultural differences and priority clashes than a project within either of two collaborating departments would do.

4. Summary

After a company merger, an organization typically wants to integrate its software tools. In this paper, we investigated a case study illustrating how this can be done, and pointed out some key features of such a process that can be summarized as early meetings, several-phase process, user involvement, separating stakeholders, active upper management, architecture-centric process, different people, and not least: it will take time.

5. References

[1] Boehm B., *Spiral Development: Experience, Principles and Refinements*, report Special Report CMU/SEI-2000-SR-008, Carnegie Mellon Software Engineering Institute, 2000.

[2] Carmel E., *Global Software Teams - Collaborating Across Borders and Time Zones*, ISBN 0-13-924218-X, Prentice-Hall, 1999.

[3] Clements P., Bachmann F., Bass L., Garlan D., Ivers J., Little R., Nord R., and Stafford J., *Documenting Software Architectures: Views and Beyond*, SEI Series in Software Engineering, ISBN 0201703726, Addison-Wesley, 2002.

[4] Kotonya G. and Sommerville I., *Requirements Engineering: Processes and Techniques*, ISBN 0471972088, John Wiley & Sons, 1998.

[5] Land R., "Applying the IEEE 1471-2000 Recommended Practice to a Software Integration Project", In *Proceedings of International Conference on Software Engineering Research and Practice (SERP'03)*, CSREA Press, 2003.

[6] Land R. and Crnkovic I., "Software Systems Integration and Architectural Analysis – A Case Study", In *Proceedings of International Conference on Software Maintenance (ICSM)*, IEEE, 2003.

[7] Linthicum D. S., *Enterprise Application Integration*, Addison-Wesley Information Technology Series, ISBN 0201615835, Addison-Wesley, 1999.

[8] Paulish D., *Architecture-Centric Software Project Management: A Practical Guide*, SEI Series in Software Engineering, ISBN 0-201-73409-5, Addison-Wesley, 2002.