Decentralized Building of Complex Software Systems: an eCommerce Case Study *

KASSEL Stephan, WINKELMANN Sabine, TITTMANN Claudia
(Westsaxonian University of Applied Sciences, 08012 Zwickau, Germany)

Abstract: Software developpment skills are only partially trained with traditional programming languages and software engineering courses. To achieve practical skills needed in software industry, complex examples are necessary which have to be generated in a project-like setting. This includes common solution finding, intensive communication and structured programming tasks. Here, a case study from the domain of eCommerce is presented, showing the complexity needed for these learning processes.

Key words: decentralised software design; software as interoperating services; web services; communication in software developpment; multi-agent systems; case study; eCommerce
CLC number: TP31 Document code: A Article ID: 0529-6579 (2007) S2-0041-05

1 Introduction

Universities of Applied Sciences play a special role in the German education systems. They have to provide an industry-oriented education including internships (for one semester) and assuring the appropriateness of their methods for industry purposes. So they connect fundamental research with practical methods. For this reason, professors are favoured who have worked in industry for several years besides their academic proficiency.

Information systems play a special role in the German university degrees since almost 20 years now. They have to combine knowledge from computer science with knowledge from business administration and economy. Using their view of both "worlds", they communicate with the business people on their requirements for software systems, and they can model and implement these requirements together with programmers having a strong background in computer science.

In the curriculum of information systems in Germany, software education plays a prominent role. But the software education consists of programming courses and software engineering as well as of business information systems and information management. Thus, programming languages and gaining experience in programming has only a very limited room, which has to be filled wisely.

Traditional introductory computer science courses often start with learning programming principles in conjunction with a popular programming language. Often, these courses are illustrated with some very simple programs exemplifying main topics. Caused by the introduction of a first programming language, most practical appointments for the students are concentrating on very simple algorithms and programs as well. The situation is getting more problematic by the growing programming language size (including in the count not the pure number of reserved words of the programming language, but taking into account the number of predefined software functions, and the number of libraries). This leads to the impossibility to teach all these concepts of the up-to-date programming languages to the students. But even teaching the programming languages well, does only a few help when dealing with enterprise software, consisting of more advanced features. For example the understanding of basic Java as a programming language is giving a very limited help for the understanding of J2EE concepts. So, most programming courses with Java try to come to the point for at least including a very limited number of J2EE lectures, to help the students grasp some basic concepts for building enterprise software.

Lately, this principle has been questioned[1], and the importance of confronting the students from the very beginning with complex examples has been recognized. This resembles the tendency of software industry in building up software systems of growing complexity, where the role of the programmer lies not in building some independent program, but in changing and en-

*收稿日期: 2007-10-28
作者简介: KASSEL Stephan, male, professor; E-mail: Stephan.Kassel@fh-zwickau.de

hancing of software modules, which are already in use.

Another way to deal with the lack of proficiency which can be gained in some concrete programming languages lies in the concentration on software engineering principles, to provide the students with some fundamental software design strategies. This emphasizes the importance of the early stages for the success of software development. In the last years, several important modelling languages for the purpose of grasping the requirements of the software to be built have become very popular. In the Unified Modeling Language (UML)[2], use cases help to provide a concise and well understandable description of the expected system behaviour. To exactly depict business processes, which have to be supported by the information system to be built, activity diagrams are well suited. They can be seamlessly integrated into the process of designing the software system. With the Architecture for Integrated Information Systems (ARIS)[3], there exists another systematic approach to build information systems, which is very popular in Germany. Especially Event-driven Process Chains (EPCs) are well used in software industry, and are providing a common understanding of the business processes, which can be used by business analysts (to change and optimize the business processes) as well as by software engineers to describe the requirements for software systems to take part in this processes. For several standard software packages, the business processes which are the fundaments of the software are described with EPCs. (As the most famous example, SAP has been completely modelled with EPCs.) The deep knowledge of one (or better several) of those modelling languages is inevitable for dealing with industry-size software systems.

To reduce the complexity of the software, several software construction principles have been deeply investigated in the last years. Some of them have found their way to industrial software providers, forming the latest buzzwords of the industry. The principles are quite simple, though the usage of the industrial solutions is partly (still) very tedious. Design patterns[4], or the Model-View-Controller (MVC) architecture[5] are very prominent ways to build complex software with clear responsibilities of the modules. Other research disciplines like Aspect-Oriented Programming (AOP)[6] are still in their rising and will play an even more important role in the future.

From the research in distributed artificial intelligence, multi-agent systems[7] are delivering several interesting ideas, which can be used to reduce complexity of software systems in an elegant way. Especially the research on self-organizing intelligent systems[8] can provide important instruments to emerge complex behaviour out of very simple individual agents with different roles in the overall system. Already the metaphor of agents helps in building complex software systems of individual components with a very limited complexity.

Using this metaphor is leveraged lately by the advent of web-services[9] and service-oriented architectures[10], providing an elegant way of interoperation of program modules, which can be originally designed to work independently. This reduces the complexity of the program modules vitally. In fact interoperability of software systems and services is a promising way of reducing the overall system complexity for software developers, allowing them to concentrate on relatively limited software modules.

In the following paragraphs, a case study on implementing a complex system with this simple metaphor is described, and some careful results of this case study are drawn.

2 Teaching Software Development for Information Systems

In the Westsaxonian University there are different study programs for information systems. In the faculty for economics and business administration we have traditionally two programs involving informations systems. In the direct study program, information systems are a choosable major field of study, which can be selected by students of business administration, public administration and business and engineering. In the postgraduate part-time studies, we directly provide a diploma for information systems, consisting of more courses in the area of computing. In both course programs, the students are taught the principles of programming languages with an integrated C++/Java programming course.

At the end of the study program, both groups have to work in a project including programming some practical modules. This project should provide the students with some additional skills they need for their later work in industry.

Resulting from the ideas sketched in the introduction section, the settings for this project have been choosen as following:

- The students have to work after the agent-oriented metaphor of building independent software modules running as independent processes.

- Interaction of the agents is done only via messages passed by web services, thus restricting the interaction to service calls provided by some other agents.

The main advantages of these settings are:

- The individual agents can be designed and implemented independent from the other modules.
- The students can do far parts of their work on an individual base, leading to individual examining the results and individually rating the students works.
- The students have to concentrate as well on the description of the interfaces they provide to the other students and on the interfaces they need to be provided by others, leading to negotiations and to writing contracts on the behaviour of their modules.
- The students evolve their negotiation skills, because a lack of negotiations leads to significant additional working loads.

3 The Case Study: Implementing an eCommerce System

Electronic Commerce (or eCommerce) plays an important role to conducting business in the last years[11]. The main task of eCommerce systems is to leverage commercial transactions via internet channels, especially via World Wide Web. The success of eCommerce depends on the business model (including the assortment, the presentation of the products, the pricing, the delivery conditions, and the services provided) of the seller. These factors make eCommerce an interesting domain for information systems with a high practical relevance.

Basically, an eCommerce system has some basic constituents, providing some business processes with relevance to the business success. This is an ideal setting for the software project which has to be conducted by the students. The task of building an eCommerce system was reduced to the main parts of the eCommerce system. Each of these parts had to be programmed individually. In both groups, the number of students involved was nearly identical and rather low (7 students in the case of the direct study program and 9 students in the case of the postgraduate part-time study program), so each student could work on his/her own.

The assignment for the students consisted of six parts, each of them being respected in the overall grading of the course.

a. The students had to model the processes being relevant for their individual tasks (sometimes they only had to model a partial process, which is part of a greater process) and they had to combine their parts to form some overall business processes, like a customer registering to the eCommerce system, buying some goods and then being delivered from the company.

b. The interoperation of the processes had to be defined by negotiation between the students. (Some of the necessary interoperations were roughly sketched by the lecturers to set a starting point for the negotiations.)

c. For their part of the work, the students have to build their own classes and a data base consisting of the relevant data they need to perform their tasks well. The data bases were deliberately not integrated to allow for more complex business processes like some individual logistics companies delivering wares on behalf of the retailers.

d. The students had to define and implement web services serving as the programmed realization of the interaction of the processes. These services had to be optimized to restrict the data exchange between different individual programs. These web services definitions resulted directly from the task figured in part b.

e. Each student had to implement some dynamic web pages to interact with the users involved in the processes. (A design styleguide was made by the students and agreed commonly.)

f. The integration of the overall eCommerce system was done in a last step, taking into consideration, that the interoperation of the system parts was already defined well during the previous steps.

The programming tasks were as follows:

- Assortment management, including definition and change of products to be sold, a list view of several products, and a detailed view of one product.
- Category management, consisting of definition and change of (hierarchical) catalog categories, definition of product types (to classify products), managing the category assignment of the products, and delivering the browsing of categories to the customers of the eCommerce system.
- Customer management, consisting of definition and change of customer data, including the login handling of the eCommerce system.

◦ Price and discount management, consisting of defining the prices for products, defining customer group specific discounts, and handling special offerings (as marketing campaigns).

◦ Basket and order management, consisting of creation and manipulation of baskets, adding products to baskets, and changing the baskets to orders.

◦ Logistics management, including stocks management, and simulating delivery as well as reordering of products.

◦ Accounting management, including outstanding receivables, and definition of customer groups.

Together, these modules provided a complete even though simple eCommerce system, which could serve as a working prototype.

4 Results and Conclusions

From the point of view of teaching industry-oriented software development, the case study showed some remarkable results. Especially the very similar setting in two different study programs exhibited interesting effects.

In the first group, the students had less experience with programming. Only one student had done more than the basic course program in programming. In the second group, the part time students had very different knowledge about programming. About $1/3^{rd}$ worked in software industry as programmers, and the other $2/3^{rd}$ had no significant experience before the study program.

The main difference consisted of the cooperation of the students. In the part time program, the cooperation was very loose, even in the common meetings with the lecturer (which had the same amount of time) the students had only a very restricted communication. Each student tried to solve their problems alone. In the first group, the students had formed a more integrated cooperation, leading to significant common solving problems with the programming environment and with the implementation of the web services (which was done in MS Visual Studio®).

Another difference was the choice of programming languages. In the first group, the students agreed on using Microsoft Visual Basic for the implementation of the system. In the second group, the students choose different programming languages (Visual Basic and Java), resulting from their programming experience. This resulted in some serious problems with incompatible definitions of web services from Microsoft and Sun, especially with complex object types as arguments for service calls. (These should be solved in the newest versions of their products.)

Despite the commonalities of the two case studies, the results were quite different.

(1) The quality of the overall solution was highly different. Although the programming experience and the programming language knowledge of the first group has been lower than the experience and the knowledge of the second group, they managed to present a better solution which was working well. In the second group, only one part was done very well. (This was the part done by a full time Java programmer who managed to do his assignment very well.) Overall, the solution of the second group had serious problems to work.

(2) The cooperation in the first team was very well, leading to an increased quality of their solution over the first initial ideas that were presented to the lecturer. Caused by an intense communication and the need of all students to get accustomed to the integrated development environment, the students helped each other to elaborate their packages.

(3) Due to the much better quality, the project of the first group had better results for the individual parts as well as for the integrational aspects. Only one part of the overall system was hardly acceptable leading to one lower grade. The other parts were well done.

(4) The individual evaluation of the lessons learned in the project was very high in the first team. They did a harder work, but they enjoyed well finishing the task and were proud of their combined work. None of the students did believe the project to be successful in the beginning. But together they managed to develop a complex software system.

(5) The high encapsulation of the project, resulting in different software systems with a strongly restricted interaction, helped to gain advantage over an integrated system. Each participant could work on his own solution, but they had to agree on the interoperability of their software.

(6) From the perspective of the lecturer, the students of the first group gained some valuable skills from doing this project for working in software industry. They learned to do substantial communication, they

learned to negotiate about the interfaces with other system parts, they concentrated first on the business processes and did the coding work after the processes were clearly defined, and they individually dealt with very simple problems, leading to simple implementations, which showed their advantage in the combination into the overall eCommerce system.

(7) The low communication and low cooperation of the second team led to bad overall results. The students did not manage to build an overall working system, substantial parts of the system have not been finished in time, the learning curve was very low, and also the impressions of the students about their own learning in the project were bad. Although their overall knowledge was much higher than in the first group, they had problems to leverage this knowledge to all team members. The results were similar to experiences of the first author of this paper from software industry, the lack of direct communication in a software project leads to significant project risks.

References:

[1] KÖLLING M, ROSENBERG J. Guidelines for Teaching Object Orientation with Java [C]. Proceedings of the 6th conference on Information Technology in Computer Science Education (ITiCSE 2001), Canterbury, 2001.

[2] ISO/IEC 19501, UML 1.4.2 specification, downloadable [EB/OL]. http://www.omg.org/docs/formal/05-04-01.pdf

[3] SCHEER A W. ARIS-Business Process Modeling [M]. Berlin: Springer, 2000.

[4] GAMMA E, HELM R, JOHNSON R, et al. Design Patterns: Elements of reusable object-oriented software [C]. Addison Wesley Professional Computing Series, 1995.

[5] DEACON J. Model-view-controller (MVC) architecture [EB/OL]. http://www.jdl.co.uk/briefings/MVC.pdf, 2000.

[6] WAND M, KICZALES G, DUTCHYN C. A semantics for advice and dynamic join points in aspect-oriented programming [C]. ACM Transactions on Programming Languages and Systems (TOPLAS), 2004, 26(5): 890—910.

[7] BERGENTI F, POGGI A. A development toolkit to realize autonomous and interoperable agents: Proc. of the fifth international conference on Autonomous agents [C]. Canada: Montreal, 2001: 632—639.

[8] DI MARZO SERUGENDO G, GLEIZES M P, KARAGEORGOS A. Self-organization in multi-agent systems [J]. The Knowledge Engineering Review, 2005, 20(2): 165—189.

[9] DAVIES N J, FENSEL D, RICHARDSON M. The future of web services [J]. BT Technology Journal, 2004, 22(1): 118—130.

[10] PAPAZOGLOU M P, HEUVEL W J. Service oriented architectures: approaches, technologies and research issues [J]. The VLDB Journal — The International Journal on Very Large Data Bases, 2007, 16(3): 389—415.

[11] FAVIER J. Europes eCommerce Forecast: 2006 To 2011, Forrester Research [EB/OL]. http://www.forrester.com/Research/Document/Excerpt/0,7211,38297,00.html, 2006.