Appears in: Proc. of the 3rd Int. Conf. on eXtreme Programming and Agile Processes in Software Engineering (XP2002), Alghero, Sardinia, Italy

Agile Development of a Clinical Multi-Agent System: An Extreme Programming Case Study

Holger Knublauch
Research Institute for Applied Knowledge Processing (FAW)
Helmholtzstr. 16
89081 Ulm, Germany
+49-731 501 8918
Holger.Knublauch@faw.uni-ulm.de

Holger Koeth
Institute for Psychology and Ergonomics, TU Berlin
Steinplatz 1
10623 Berlin, Germany
+49-30 314 79510
hko@awb.tu-berlin.de

Thomas Rose
Research Institute for Applied Knowledge Processing (FAW)
Helmholtzstr. 16
89081 Ulm, Germany
+49-731 501 520
Thomas.Rose@faw.uni-ulm.de

ABSTRACT
This paper presents an agile development methodology for multi-agent systems based on XP and reports on our experiences with it for the implementation of agents that support the clinical information flow. Our case study shows that XP is a very promising approach for agent development, because agents are rather small units that can be implemented, tested and refactored rapidly without risking the overall system's functionality.

Keywords
Extreme Programming, Multi-Agent Systems, Case Study

1 INTRODUCTION

An *agent* is an encapsulated computer system that is situated in some environment, and that is capable of flexible and autonomous action in order to meet the objectives of its principal [3]. In domains like clinical information systems, agents have a huge potential to support human principals by providing intelligent information services, e.g. by delivering critical patient data to the responsible physician proactively at the right time.

The agents' flexibility and their individual, autonomous viewpoints make multi-agent systems difficult to build. The emerging discipline of *Agent-Oriented Software Engineering (AOSE)* [2] aims at defining systematic methodologies which guide the developers through rather waterfall-based phases such as role analysis and services design. As a result, the existing AOSE approaches are expensive when requirements are weakly specified. When agents are to be introduced into an existing clinical workflow, clinical staff must be closely integrated into requirements elicitation and agent design. Since AOSE methods rely on rather formal modeling artifacts and deliver feedback late, they appear to be inappropriate for this collaboration.

2 XP OF MULTI-AGENT SYSTEMS

We have introduced a novel, agile methodology [4], which optimizes the practices of *Extreme Programming (XP)* [1] for the development of multi-agent systems. In our approach, clinical experts use a simple process modeling tool to capture the existing clinical activities and the data flow between them. These models are fed into an XP cycle in which the processes are "agentified" by incrementally delegating activities to agents and by digitizing the data flow. The resulting process models can be regarded as XP story cards which define the services and the input and output data of each agent type (see [4] for details).

We have used our approach to develop a prototypical clinical multi-agent system in Java. Our case study was conducted as an XP course for Computer Science students at the University of Ulm, Germany in 2001 (we will conduct a second course in April, 2002). The course took place in a single office with 5 PCs. It involved 8 students, a coach (the first author), and a medical doctor (the second author), who was permanently on-site to provide clinical knowledge. The course took 7 days, the first two of which were used to introduce the students to XP, agents, and the tools (IntelliJ, JUnit and CVS).

40-hour-week. The practical work itself was done during one 40-hour week. The students were explicitly not encouraged to work overtime. After the course, the students reported that they used to be quite exhausted after a full day of pair programming, but were very disciplined and concentrated while in the office. Nevertheless, the atmosphere was very relaxed and enjoyable and thus stimulated creativity and open, honest communication. This helped to prevent communication barriers between technicians and the clinical expert.

Planning game. At the beginning of each day, the team jointly defined the features that were to be implemented next. Since the process models (story cards) described the phases of a patient's treatment on her way through the hospital in a rather sequential style, we found it most useful to implement the agents in their order of appearance within the process. We locally focused on those agents that – according to the domain expert – promised the most business value.

Pair programming. Each pair of programmers had to develop and test their individual agent in isolation. The students found pair programming very enjoyable and productive. The intense communication helped to spread a basic understanding of the clinical processes among the programmers. We changed pairs almost every day.

Testing. Agents are typically rather small and loosely-coupled systems which solve their tasks in relative autonomy. As a consequence, writing automated test cases is quite easy for agents, because the single agents have a small, finite number of interaction channels with external system units. Many tests therefore consisted of sending a test message to the agent and of checking whether the expected reply message was delivered back and whether the agent's state has changed as expected. The students found testing quite useful to clarify requirements although it was considered to be additional work by some. During the course, the students have implemented 76 test cases, amounting to 4909 lines of code, while the 43 agent source code classes amount to 5673 lines. The students enjoyed using JUnit very much, because correct test runs improved motivation and trust in the code. We found specifying and implementing tests extremely important, because it clarified several misunderstandings between the domain world and the programmers.

Collective ownership. Since each pair only operated on the source package of a single agent, there was barely any overlapping. Only ontology classes (which describe the content of agent messages) had to be modified by various teams. Coordination of these changes was accomplished very informally by voice and the CVS.

Coding standards. In the beginning of the project, we defined a project-wide coding standard that was very easy to follow, because the Java tool we used provides automated code layout features. Thus it was very simple to shift implementation tasks between the pairs and to change pair members regularly.

Simple design. The students were explicitly asked to focus on programming speed instead of comprehensive up-front designs. This seemed to be sufficient because the agents were rather small units with few types of tasks to solve. Despite the focus on simplicity, experienced students almost automatically identified some useful generalizations of agent functionality. Our initial process model underwent several evolutionary changes. Despite the various small changes, the overall design remained quite stable throughout the project, so that our simple process modeling framework proved to be appropriate.

Refactoring. Since the agents were rather small units, they were very easy to maintain and refactor. Even if an agent evolved into a performance or quality bottleneck after a series of refactorings, it was possible to completely rewrite it from the scratch without risking the functionality of the overall system. IntelliJ's refactoring support was valuable.

Continuous integration and short releases. The agents were uploaded onto the CVS server and integrated at least every evening. Since the students were only allowed to upload those agents that passed all test cases, there were almost no integration problems. Agent interactions were tested and presented on a beamer with the help of a small simulation environment that could trigger external events.

On-site customer. In the questionnaires that were filled out by the students after the course, the presence of the domain expert was very positively evaluated. He was asked to provide clinical knowledge regularly, at least once an hour, so that expensive design mistakes were prevented. His presence did not even mean an overhead for him, because he could use the "spare time" for other types of work on his own laptop.

Metaphor. Many communication bottlenecks and misunderstandings between clinicians and developers are due to different terminology and comprehension. Metaphors, which map clinical domain concepts onto symbols the engineers are acquainted with, can help. For example, the process of anesthesia, with its induction, monitoring, and extubation phases, can be compared to aviation, where take-off, cruising, and landing are the main activities. This metaphor helped us to draw some insightful parallels between the requirements of clinical monitoring devices and cockpit technology.

3 RESULTS

Our case study indicates that XP can be a very natural approach for the design and implementation of multi-agent systems. Since the complex interaction scenarios and emerging behaviors between agents make pre-planning very difficult, the evolutionary practices of XP appear to be a better choice than conventional engineering approaches. Particularly the close involvement of domain experts simplifies the matching between agent services and the requirements of the human actors that are supported by agents. Since agents are rather small, autonomous units, they can be implemented, tested and refactored rapidly without risking the overall system's functionality. Last but not least, our project has shown that XP is very enjoyable and motivating for the developers and domain experts.

REFERENCES

1. Beck, K. *Extreme Programming Explained: Embrace Change*. (Addison-Wesley, 1999)
2. Ciancarini, P. and Wooldridge, M., editors. *Agent-Oriented Software Engineering* (Springer-Verlag, Berlin, Heidelberg, New York, 2001)
3. Jennings, N., Sycara, K., and Wooldridge, M. A Roadmap of Agent Research and Development. *Int. Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7-38 (1998)
4. Knublauch, H. Extreme Programming of Multi-Agent Systems. *Proc. of the First Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Bologna, Italy (2002).