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

Extreme Programming of Knowledge-Based Systems

Holger Knublauch
Research Institute for Applied Knowledge Processing (FAW)
Helmholtzstr. 16, 89081 Ulm, Germany
+49-731 501 8918
holger@knublauch.com

ABSTRACT
For most knowledge-based systems, knowledge must necessarily be modeled evolutionary, in a close collaboration between domain experts and engineers. While the Knowledge Engineering literature suggests to follow rather waterfall-based approaches, we argue that agile development methodologies like Extreme Programming have a huge potential to thrive and prosper in domains like knowledge-based systems. Agile approaches are optimized for projects with frequently changing requirements, provide explicit support for collaboration and rely on a minimum of modeling artifacts with smooth transitions between them.

Keywords
Agile methodologies, knowledge-based systems, ontologies

1 INTRODUCTION

Since *knowledge-based systems (KBSs)* are software artifacts that use an explicit, formal model of human expertise, their construction requires the close involvement of domain experts into the process. Research in *Knowledge Engineering (KE)* aims at defining methodologies that allow to construct KBSs in a systematic and controllable manner. However, although KE approaches are rather waterfall-based, they provide little support for the transitions between knowledge, knowledge models, and the KBS's remaining artifacts [1]. We argue that such heavy-weight methodologies are often not suitable for KBSs, because they induce high costs of change and do not fully exploit the creative potential of collaboration. Instead, we explore the application of agile approaches like Extreme Programming (XP), which are geared for frequently changing requirements and models, for the price of putting constraints on the size of the developing team. A major goal of this paper is to increase the awareness that XP is highly relevant for KBS development – and vice-versa.

2 THE NEED FOR COLLABORATION

Any non-trivial KBS development process will involve people that take one of the following three roles. *Domain experts* (e.g., clinicians) possess the domain expertise needed for building, verifying, and validating knowledge models. *System developers* build the overall software system, including reasoning methods. *Knowledge engineers* mediate between the informal domain world and the system's formal requirements. Each of these groups has different logics and attitudes (cf. [5]). For example, domain experts are usually oriented towards the individual case while knowledge engineers try to identify global solutions. Complex and highly specialized domains such as medicine are further characterized by a distribution of knowledge between several domain experts (e.g., surgeons and anesthetists). Development methodologies must reflect these different logics and individual view points by appropriate languages and tools. Additionally, they must not neglect human factors, because experience shows that the bottleneck of building knowledge models lies more in the social process than in the technology (cf. [2]). The construction of a functioning collaborative network between the developers is needed because "Knowledge is commonly socially constructed." [6]

3 THE NEED FOR EVOLUTION

Human cognition and scientific theory construction are iterative processes. Cognition is based on the construction of theoretical models that are exposed to experimental data from real or simulated worlds. Knowledge modeling is a kind of theory construction in which human experts construct formal theories about a domain and expose the resulting knowledge models to real or simulated worlds. Tests in both worlds produce feedback which allows to revise the knowledge models. When installed in the application scenario, the system even changes the real world and thus produces new requirements, which recursively suggest changes to the knowledge model.

There are other reasons why knowledge models will almost necessarily evolve during KBS development (cf. [5]). (1) Finding requirements is hard, because the potential users are often unable to assess the benefits or usage scenarios of the new system, and because the system modifies the work processes in which it is installed. (2) The knowledge acquisition process itself can not be completely planned, because the various developers and groups involved in the process face each other with different and unknown cognitive and social perspectives. (3) Knowledge models are often based on wrong assumptions, because knowledge modeling requires the domain experts to transparently expose their daily practice, but this practice necessarily operates with deception. (4) Knowledge – especially in non-deterministic domains such as medicine – is inherently vague. For the new medium, knowledge is being translated and reorganized and evolves in the process of being encoded and formatted for the system.

4 EXTREME PROGRAMMING OF KBS

Although XP is being widely used among mainstream software developers, its ideas have not been transferred into the KBS community yet. In the following subsections, we therefore present examples of how to adapt the values, principles, and practices of XP to KBSs. We have applied these successfully for the development of a knowledge-based multi-agent system [3] and other KBSs. Note that the size of this paper prevents us from going into details.

Simplicity. We suggest to start with knowledge models that are as simple as possible, and refine and evolve them later when the requirements are better understood. Knowledge models should only represent what is needed to solve the given tasks, i.e. modeling should be driven by the model's purpose. Simple mechanisms such as *Ripple-Down-Rules*, where knowledge modeling starts with simple rules which are refined iteratively, do not require a comprehensive analysis phase. The value of simplicity can also be applied to the choice of a language for the knowledge metamodels (ontologies). Formal ontology languages from KE are relatively difficult to learn and apply. Since industrial object-oriented languages feature most of the expressive elements found in frame-based KE languages (namely classes, attributes, and relations), they represent a simple alternative to formal ontology languages and enable the developers to follow the XP principle of *Traveling Light* to reduce the cost of change and model transitions [4].

Communication. Most of the bottlenecks in KBS development are concerned with the definition of suitable ontologies [2]. Ontologies are the main link between the domain world and the system world, which have partially opposing requirements. Ontologies should be built in a close collaboration between domain experts and knowledge engineers (cf. [2]). Ontologies will usually evolve, in particular when the domain experts have gathered experience in building knowledge bases on top of them, but in our projects we found that the collaboration between people with as different view points as programmers and anesthetists can lead to surprisingly stable metamodels. The close collaboration between experts and engineers also promotes learning from and teaching each other. In contrast to ontologies (classes), the knowledge bases (instances) can usually be modeled by the domain experts alone, without assistance by engineers. Following the very idea of XP, domain experts should model their knowledge in pairs. *Pair Modeling* can significantly improve model quality, since tacit knowledge and personal preferences are generalized, so that the domain experts' logic approaches the engineers' logic. An important communication aspect is *Humility*, because the collaboration of very different groups of people in a KBS project enforces a different code of ethics than "normal" software development projects.

Refactoring and Design Patterns. The XP practice of *Refactoring* (i.e., improving the design of existing code without breaking its functionality) can be applied to knowledge modeling as well. On ontology (class) level, refactoring means to apply the well-known refactoring patterns from object-oriented programming. On knowledge base (instance) level, refactoring can mean to combine duplicate model elements, to improve visual models (e.g., by rearranging nodes in a graph), or to extend the documentation of elements that have proven to be stable for various releases. The *Theory Refinement* community has produced various approaches on how knowledge models can be systematically simplified. The common goal of all these activities is to keep the system as simple and maintainable as possible. Another important application scenario of refactoring is the generalization of existing knowledge models for future reuse. Further potential for improving development efficiency lies in the application of object-oriented *Design Patterns* to knowledge modeling. For example, the *Composite* pattern is applicable to many ontologies in which domain concepts form a hierarchy. Annotating design decisions with Design Patterns improves communication between modelers and support reuse.

5 CONCLUSIONS

In this document, we have shown that evolution and collaboration are key requirements in KBS development. While these requirements are insufficiently supported by conventional KE methodologies, the good news for the agile community is that the seeds of XP have a huge potential to thrive and prosper in the KBS development.

REFERENCES

1. Benjamins, R., Fensel, D., Pierret-Golbreich, C., Motta, E., Studer, R., Wielinga, B., and Rousset, M.-C. Making knowledge engineering technology work. *Proc. 9th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, Madrid, Spain (1997)
2. Holsapple, C., and Joshi, K. A collaborative approach to ontology design. *Communications of the ACM*, 45(2), 42-47 (2002)
3. Knublauch, H. Extreme Programming of Multi-Agent Systems. *Proc. Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Bologna (2002)
4. Knublauch, H., and Rose, T. Round-trip engineering of ontologies for knowledge-based systems. *Proc. 12th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, Chicago, IL (2000)
5. Rammert, W., Schlese, M., Wagner, G., Wehner, J., and Weingarten, R.. *Wissensmaschinen: Soziale Konstruktion eines technischen Mediums. Das Beispiel Expertensysteme* (Campus Verlag, Frankfurt, 1998)
6. Salomon, G. *Distributed Cognitions: Psychological and educational considerations* (Cambridge University Press, Cambridge, UK, 1993)