THE AGILE DEVELOPMENT OF RULE BASES

Abstract: Recently, with the large scale practical use of business rule systems and the interest of the Semantic Web community in rule languages, there is an increasing need for methods and tools supporting the development of rule based systems. Existing methodologies fail to address the challenges posed by modern development processes in these areas: namely the increasing number of end user programmers and the increasing interest in iterative methods. To address these challenges we propose and discuss the adoption of agile methods for the developments of rule based systems. The main contribution of this paper are three development principles for and changes to the XP development process to make it suitable for the development of rule based systems.

Keywords: Software Engineering, Knowledge Engineering, Rule Bases, Agile Methods, Knowledge Based Systems

Number of words: 4499 words

Type of submission: (please select)

[X] Full academic paper
[ ] Position paper
[ ] Practice-based experience report
[ ] Work-in-Progress
[ ] Other: please specify

Track: (please select)

[ ] Managing IS Development
[ ] Innovation in Information Systems Development
[ ] Enterprise Systems Development & Adoption
[ ] Public Information Systems Development
[ ] Agile and High-Speed Systems Development Methods
[ ] Information Systems Engineering
[ ] Business Systems Analysis & Design
[X] Data and Information Modelling
[ ] ISD Education
[ ] ISD in Developing Nations
[ ] Legal and Administrative Aspects of ISD
[ ] Service Oriented Modelling and Semantic Web Technologies

THE AGILE DEVELOPMENT OF RULE BASES

1. INTRODUCTION

Recently, with the large scale practical use of business rule systems (Seer 2005) and the interest of the Semantic Web community in rule languages (Kifer 2005) there is an increasing need for methods and tools supporting the development of rule based systems.
There already exists a large body of research into this topic, but most of it is older and does not account for (1) changes in the training of the developers of such systems (2) the shifting nature of project that create rule based systems.

Changes in programmers training

In the early days of rule based systems, these systems were created by a small number of highly trained knowledge engineers that could easily work with logic and highly formal models. Many methodologies and tools still assume this kind of user. These days, however, software artefacts are often created by end user programmers – people trained for a non-programming area that just need a program, script of spreadsheet as a tool for some task; in fact end user programmers are estimated to outnumber professional programmers by four to one (Scaffidi 2005). End user programmers are particularly important for the Semantic Web and the business rule community:

- Business rules are usually small parts of larger systems that allow customizing of an application for a particular company/organisation. Business rules are used to represent that part of business applications that changes in time or from company to company. The prospect of business rules is to allow the quick adaptation of applications to (changing) business needs. In order to fully realize this prospect business rules need to be changeable by the same people that use them and hence often end user programmers.
- The Semantic Web is a candidate for the next development stage of the world wide web and it will have to be build largely by the same people that build and maintain the current web – a high percentage of these people is known to be end user programmers (Rossen 2004) (Harrison 2004)

Changes in project structure

Current best practices stipulates that rule bases form only a part of a larger application - only that part concerned with changing or complex logical relations (Merrit 2004). This tends to result in smaller rule bases. Also, the fact that end user programmers are expected to maintain rule sets means that these have to remain relatively small.
Recent years also saw the increasing adoption and acceptance of iterative and evolutionary developments methodologies (Larman 2003) – they are now widely believed to be superior to waterfall like models (MacCormack 2001). This development too, is not reflected in most current methodologies for the development of rule based systems.

Overview

Current methodologies for the development of rule based systems do not adequately address the challenges posed by end user programmers and smaller projects; they also often do not take advantage of the ideas of iterative application development.
To address this shortcoming we propose an adoption of the ideas of agile software development for the development of rule bases. This paper starts with a short overview of the

core principles of agile software development and Extreme Programming (XP). The main part of the paper then consists of the discussion of why and how it needs to be adapted for the development of rule bases. Towards the end we shortly discuss how the agile development of rule bases compares to existing methodologies for this task.

2. AGILE SOFTWARE DEVELOPMENT AND XP

Agile software development is based on the core ideas of simplicity and speed. An agile method is one that defines software developments as (Abrahamsson 2004):

- Incremental: Software is created in an evolutionary, iterative way with many small software releases in rapid succession.
- Cooperative: Software developers and customers are constantly working together in close cooperation.
- Straightforward: The method itself is easy to understand and easy to apply.
- Adaptive: The method makes it easy to react to late braking changes.

Many agile methods for software development have been proposed, such as XP, Scrum, FDD or DSDM (Beck 1999) (Schwaber 2002) (Palmer 2002) (Stapleton 1997). The description in this paper is based on the well known XP methodology, although most of the ideas presented are applicable to the general problem of adopting agile methods to the development of rule bases. For brevity only a short overview of XP can be given here, the interested reader finds more complete descriptions in (Beck 1999) (Beck 2004).

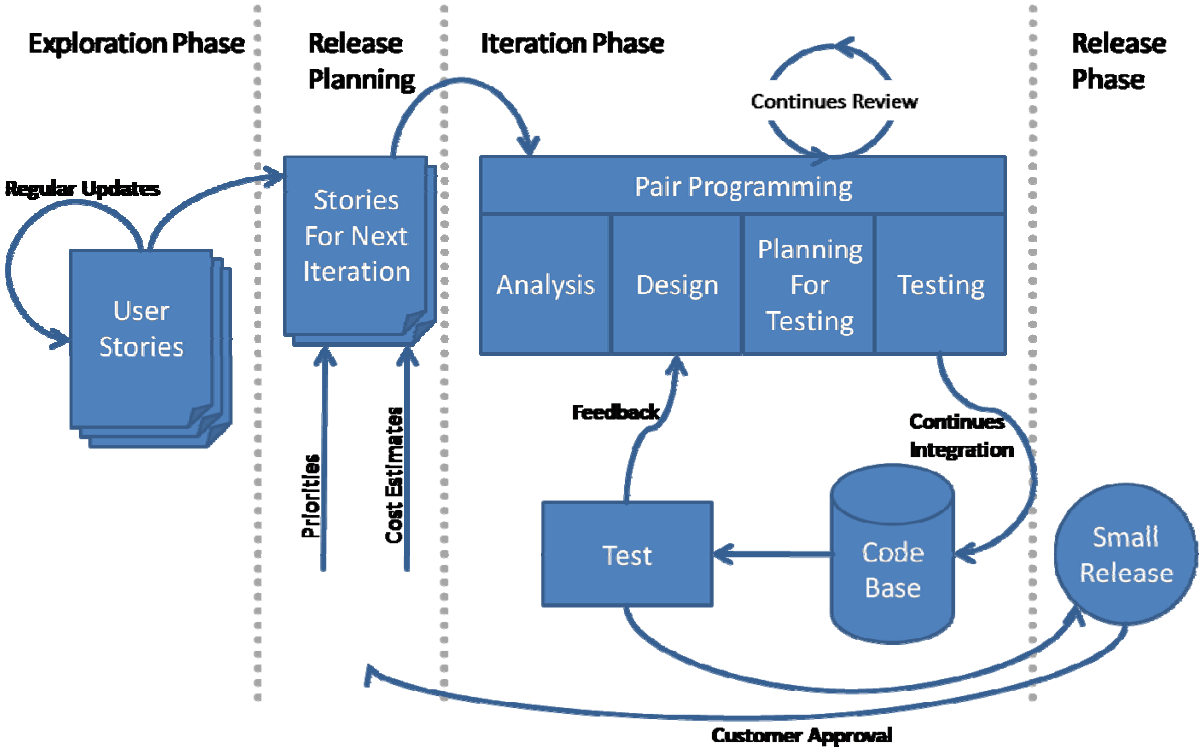

Figure 1: A simplified view of XP according to (Abrahamsson 2004). The maintenance and death phases have been omitted for brevity.

A simplified view of the XP methodology is shown in Figure 1. XP can be understood as being structured in six phases (Abrahamsson 2004):

- In the **exploration phase** the customers create the requirements for the software system by specifying "User Stories" – things the system needs to do for them. At the same time the developers familiarize themselves with the technology needed for the project and investigate any critical parts by creating prototypes.
- In the **release planning phase** the user stories are prioritized and a rough cost estimate is made for them. Also in the planning phase customers and developers agree on the user stories to be included in the first release.
- In the **iteration phase** a planned releases is first broken down into multiple iterations. Within each iteration the user stories are broken down into programming tasks. The iterations are time boxed (typically 1-3 weeks) and user stories are added or removed to arrive at a plan realizable in this time. A quick design is done at the beginning of each iteration and testing is used as a tool for verification and validation. Code is integrated continuously and run against acceptance tests representing user stories (that are ideally created directly by customers). XP also stipulates a number of best practices for development, such as stand-up-meetings, pair programming, collective code ownership or the recommendation to refactor mercilessly.
- In the **release phase** further tests are made to ensure that the system is fast and reliable enough to be deployed. Feedback from the released and deployed system influences the release planning. Work on the next release then commences.
- The **maintenance** and **death phases** not shown here deal with later development stages when further development takes second place to costumer support.

Extreme programming is a general software development methodology that can be applied to the development of rule based systems. However, XP was created from experiences mainly in object oriented programming and its application to rule based systems is not always straight forward and warrants discussion. Based on our experiences with the development of multiple rule bases we therefore propose three new principles, changes to the iteration planning phase and two new best practices for the actual development.

3. CORE PRINCIPLES

We propose three guiding principles for the development of rule bases with XP.

Program first, knowledge second

Many existing methodologies for the creation of rule bases systems put at their core the idea that rules bases are not only programs but first and foremost attempts to capture the knowledge in one domain. This idea imposes new validation criteria on the development of such systems: no longer is it possible to validate a system by showing that it completes the tasks it was created for; rather such systems must be validated by showing/proofing some mapping between the systems and the knowledge of the expert/in a domain. While unassailable from a theoretic point of view, this "knowledge first" idea causes a multitude of problems in practice:

- There is no proven method to decide when a domain is **completely** or **adequately** captured; when the modelling task is finished. This missing cut-off point can lead to the modelling process dragging on for a very long time. Some methodologies (see the related work section for details) try to assail this problem by giving excessive guidance on knowledge acquisition and proposing a multitude of interconnected formal models to identify the relevant knowledge. However, this methodologies thereby become difficult to understand and hard to implement.

- The **correctness** of the domain formalization is posing similar problems. While formal methods can be used to at least show some notion of internal consistency it is still hard to verify rule based system independently of actual tests. Methodologies that centre around the "knowledge first" idea often require long, top down modelling stages that do not give much feedback to the user.

The "Program first, knowledge second" principle was created as an answer to these problems. It states: even a rule based system is first and foremost created to solve; and evaluated against a number of tasks. The idea that it represents the knowledge in a domain is secondary to that: it is only a guideline on how this task is to be solved.

Interactive Rule Creation

End user programmers tend to build software system in a try-and-error, incremental way; learning while they are building the system (Myers 2006). Many development environments for object oriented or script languages have long offered support for this kind of working: compiling programs on the fly while they are edited, allowing starting of a program at the touch of a button and some even allowing changes to the code while a program is running. For current development environments for rule based systems similar support is not always offered. Hence the need for the "Interactive Rule Creation" principle: the agile development of rule bases must be supported by an environment that enables the developers to instantly try out their rules and see how they interact with the rest of the rule base. To try out a rule can mean both to see which (new) inferences a rule enables and to pose queries to the entire rule base. Debugging support must be available for the cases where the results do not match the user's expectations.

Explanation is Documentation

Rule based systems are often created for high level task; tasks that haven't been implemented before and for which there is no clear algorithm. With these application scenarios comes the need for explanation subsystems. It has long been realized that in order to gain acceptance for such systems they need to somehow explain the process that lead to an answer. For example a typical application for rule based systems is the rating of applicants for loans based on their credit history and other data; the decision on whether to grant or deny a loan is then based on this rating. For such systems an explanation of the reasoning process is important to gain acceptance by the users and possibly to help the user deal with an upset customer. For these reasons explanation subsystems were part of many rule based systems since the days of the early expert systems. The explanation facilities differ between systems but usually allow investigation of the proof after a result has been created. To make the proof better understandable and more informative, rules are often augmented with human readable templates, provenance information. In some systems meta-rules are used to aggregate or hide certain parts of the proof for a more concise representation.

The important observation here is that the data added to the rules for the purpose of explanation is a kind of documentation. This leads directly to the "Explanation is Documentation" principle: in rule bases that support explanations the data added for the explanation is (most of) the documentation. Explanation data and documentation is managed as one.

4. ITERATION PLANNING

XP stipulates that at the beginning of each iteration the user stories are broken down into programming tasks. This informal analysis phase needs to be adapted for rule bases systems for three reasons:

- Rule bases are often only parts of larger systems; hence the system architecture has to be used to identify what a user story means for the rule base.
- Rule bases are often build to represent complex high level knowledge that is not widely available in an organisation; hence a systematic attempt must be made to identify sources for this knowledge.
- It is prerogative for the creation of rule bases that the entities the rules work on are well defined; e.g. it causes havoc when one rule assumes employees to include retirees and another assumes the contrary.

To address these challenges we propose to prefix the decomposition of user stories into programming task with a refinement process (see Figure 2).

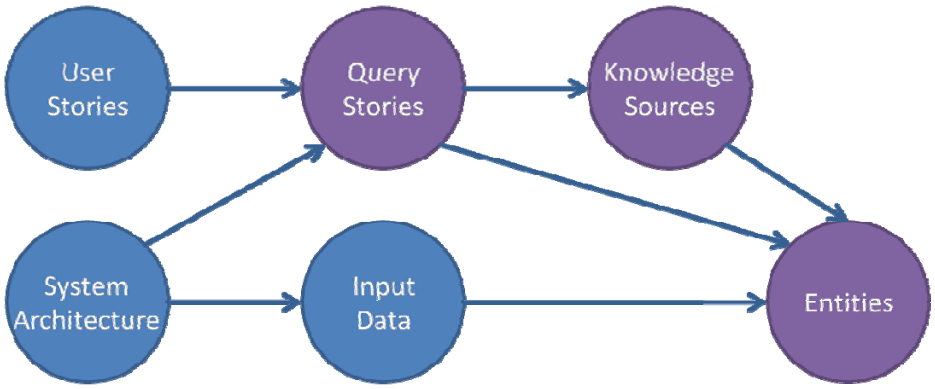

Figure 2: A high level view of the revised iteration planning stage.

The starting points are the user stories for an iteration and the system architecture – some notion of how the rule base is integrated into the overall system. Based on this information query stories are created. A query story describes what kind of query is asked to the rule base in order to enable a user story. Query stories (rather than user stories) are used to create the acceptance tests for the rule base.

In the next step the query stories are evaluated on whether the knowledge needed to implement them is available within the project group creating the system. If not, knowledge sources must be identified to fill this gap. Knowledge sources can be for example:

- Additional experts available to help.
- Documents such as text books, manuals, documented procedures etc.
- Reusable data such as web ontologies.

Parallel, based on the overall system architecture, the input data is identified. Input data is the data a rule base is meant to process, such as tables in existing databases or messages arriving from other program parts. The purpose of the collection of the input data is to use it for the identification of the entities in the domain. Other inputs for the identification of the domain entities are the query stories and the knowledge sources. The output of this process is a description of the entities the rule base should work with. This description can be informal or be a formal ontology that can be used for verification throughout the creation of the rules. In

many cases the description of the entities can be largely derived from the schema for- and the documentation of- the input data.
Query stories, knowledge sources and the identified entities form the input for the rest of the iteration that commences with the decomposition of user stories in programming tasks.

5. DEVELOPMENT

In object oriented or procedural programming the overall structure of the program is explicitly created by the developers and often directly visualized; not so in rule based systems. In object oriented programs the overall structure is given by the class hierarchy, links between objects and method calls. In rule based systems the connections between rules are made by the inference engine on an on-demand basis depending on data and the query asked. This automatic interaction between rules makes rule based systems so flexible; however, the hidden nature of the rule interactions also causes problems:

- It makes it difficult for developers to see the overall structure of the program. In particular this causes problems during refactoring where it cannot easily be seen which other rules are affected by a change.
- It makes it hard to verify that the rule interactions work as intended and complicates the diagnosis for the cases where it doesn't.

Another problem is that often rule languages do not have an equivalent for the strong typing of many programming languages and that hence less errors can be found at compile time.
These challenges mandate some additions to the methods and best practices of the actual development stage.

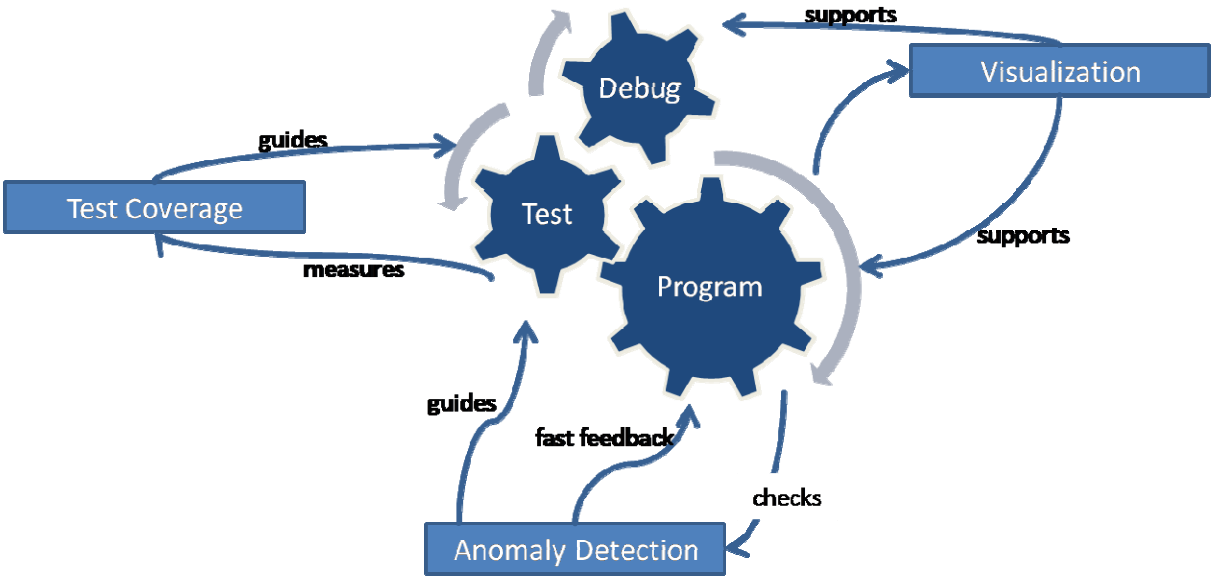

Figure 3: A high level view of the revised development stage.

A view of the development stage is shown in Figure 3. At its core it shows the test, debug and program activities directly interconnected – as mandated by the interactive principle described earlier. These activities are supported by test coverage, anomaly detection and visualization. Test coverage plays a similar rule as it does in the development of object oriented systems; visualization and anomaly detection, however, are unique addition for the development of rule bases and are discussed in the next two sections.

Anomaly Detection

Anomalies are symptoms of probably errors in the knowledge base (Preece 94). Anomaly detection is a static verification technique to identify and show these anomalies to the user. Anomalies are often concerned with the hidden interaction structure between rules – one of the most commonly identified anomalies is a rule cycle. Anomaly detection is a well established tool for the development of rule bases. Anomalies are commonly divided into four large categories (Preece 94):

- Circularity, rules that can cause inference engines to run indefinitely.
- Ambivalence, in particular contradictory rules.
- Deficiency, input data never used in rules or other symptoms of missing parts in the rule base.
- Redundancy, rules that conclude statements that are never used, duplicate rules, subsumed rules, unsatisfiable rules etc.

Anomaly Detection heuristics should be used constantly to check the rules as they are formulated. The anomalies found form a fast feedback to the developer during programming and allow to quickly correct mistakes. Anomalies in the rule base that the developer does not understand as error can form a guidance for parts of the rule base that should be thoroughly tested.

In accordance with the paradigm of "interactivity" stated earlier, the anomalies should be calculated without requiring specific attention of the user and should instantly reflect the changes done by the developers.

Visualization

The introduction of this section has established that the invisibility of the overall structure of rule bases is a major impediment for their creation. One way to alleviate this problem is to attempt to identify and visualize this structure. This can be done based on the static or the dynamic structure of the rule base (Zacharias 2006):

- The static structure is based solely on the rules (and not the facts) and tries to identify the interactions between rules that could happen, if the rule base where to be used with the right facts.
- The dynamic structure of a rule base is based on the actual rule interactions that happen during queries to the rule base.

Such visualizations that show the actual/possible interactions between rules can then be used by the developers to aid programming, debugging and in particular refactoring.

6. RELATED WORK

CommonKADS (Schreiber 1999) is to date the most influential academic methodology for the development of knowledge based systems. CommonKADS understands the development of knowledge based systems mainly as a modelling task. It emphasizes the construction of conceptual models of the knowledge, the entire system and the context of the application. CommonKADS stipulates the definition of a KBS in a series of six interconnected models: the organization, the task, agent, knowledge, communication and design model. CommonKADS claims to support the iterative development of KBS but requires a large up front analysis phase for the initial definition of these models. Because of this large, formal analysis phase it is not well suited for the kind of end user driven; smaller projects that are the topic of this paper (Kingston 1992). It imposes too much of an overhead and end user programmers with no prior experience in building KBS cannot be expected to build correct,

formal specifications; they can also not be expected to learn CML as another formal language just for the specification.
The approach to build a KBS in a top down fashion, as the stepwise refinement of interconnected models is not unique to CommonKADS and is in fact used in many methodologies (Stokes 2001)(Plant 1997)(Yen 1992), but they all suffer from the problems described above. PragmaticKADS is a variant of the initial KADS methodology that tries to addresses this problem of the large initial overhead, but it is defined only at a very high abstraction level, makes no allowance for refinement steps and has almost no discussion of validation and verification activities (Kingston 1992).
Another group of methodologies arose from the attempt to reconcile the building of KBS as a series of rapidly created prototypes with the need for formal processes to monitor and control a project's progress (Miller 1990)(Weitzel 1989)(ANSI 1992). All of these methodologies have prototypes at their centre, but also have some structure to allow monitoring and controlling. These methodologies see the different prototypes created during the development as different from the actual application (that is created by re-implementing the prototypes) and some have only a fixed number of stages. In this sense they are not real iterative methods.
Another related thread of research are methodologies for the development of ontologies, in particular the On-To-Knowledge (OTK) methodology (Sure 2002). OTK proposes a relatively lightweight, multi-stage approach that allows for some iterative cycles. However, this too is not really an iterative method since it considers the refinement steps only when a final product fails the evaluation. OTK also focuses on ontologies as tools for navigation and retrieval, less for inferences – hence there is for example no notion of "test" in the OTK methodology.

7. CONCLUSION AND FUTURE WORK

Current methodologies for the creation of rule based systems fail to address the challenges posed by changes in project structure and in developers training. Agile methods from the world of object oriented programming seem better suited to meet these challenges but cannot be applied to rule based systems without considerable changes. To facilitate the application of agile methods for the development of rule based systems we propose three new development principles and changes to the iteration planning and development stages. The three principles are "interactive rule creation", "explanation is documentation" and "program first, knowledge second". The iteration planning stage is changed to account for (1) the embedded nature of rule bases (2) the need to identify the domain entities before rules are created and (3) the problem that the knowledge needed to create a rule base may not be easily available. The development stage is extended with anomaly detection and rule base visualization to deal with the invisibility of the overall rule base structure that hinders programming, debugging and refactoring.
For the future we plan to further develop the application of agile methods for rule based systems, in particular to account more clearly for projects where developers and customers are in fact the same people. There are also still considerable technical challenges in creating anomaly detection heuristics that are fast enough to support interactive rule creation. Finally the overall visualizations of rule bases is a question that has been largely ignored by the research community and that hence knows only few approaches and implementations.

8. REFERENCES

Abrahamsson, P. & Salo, O. & Ronkainen, J. (2002) Agile software development methods Review and analysis. ISBN 951-38-6009-4

ANSI, 1992, Life cycle development of knowledge based systems using DoD-Std 2167A ANSI/AIAA G-031-1992
Beck, K. (1999) Embracing Change with Extreme Programming. *IEEE Computer* 1999:32-10, pp. 70-77
Beck, K. & Andres, C. (2004) Extreme Programming Explained. Embrace Change. ISBN 978-0321278654
Harrison, W. (2004) The dangers of end-user programming. *IEEE Software*, Juli/August:5-7
Kifer, M. & de Bruijn, J. & Boley, H. & Fensel, D (2005) A realistic architecture for the semantic web. *Proceedings of the First International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML 2005)*
Kingston, J (1992) Pragmatic KADS: a methodological approach to a small knowledge-based systems project. International Journal of Knowledge Engineering 1992:9-4, pp. 171-180
Larman, C. & Basili, V.R. (2003) Iterative and Incremental Development: A Brief History. *IEEE Computer*, June 2003:36-6, pp. 47-56
MacCormack, A. Product-Development Practices That Work *MIT Sloan Management Rev.* 2001:42-2, pp. 75-84
Merrit, D. (2004) Best Practices for Rule-Based Application Development. *The Architecture Journal*, January 2004
Miller, L. (1990) A Realistic Industrial Strength Life Cycle Model for Knowledge Based Systems Development and Testing *AAAI Workshop Notes: Validation and Verification*.
Myers, B. & Ko, A. & Burnett, M. *Invited Research Overview: End-User Programming*. ACM Conference on Human-Computer Interaction (CHI'06), 2006.
Palmer, S.R. & Felsing, J.M. (2002) A Practical Guide to Feature-Driven Development. ISBN 978-0130676153
Plant, RT & Gamble, R. (1997) A multilevel framework for KBS development International Journal of Human-Computer Studies 1997:46, pp. 523-547
Preece, A. D. & Shinghal, R. (1994) *Foundation and Application of Knowledge Base Verification. International Journal of Intelligent Systems*, 1994:9-8, pp. 683-701
Rossen, M.B. & Balling, J. & Nash, H. (2004) Everyday programming: Challenges and opportunities for informal web development. *Proceedings of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing.*
Scaffidi, C. & Shaw, M. & Myers, B. (2005) Estimating the number of end user programmers *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pp 207-214, 2005.
Schreiber G. & Akkermanns, H. & Aniewierden, A. & de Hoog, A. & Shadbolt, N. & Van de Velde, W. & Weilinger, B. (1999) Knowledge Engineering and Management: The CommonKADS Methodology. ISBN 978-0262193009
Schwaber, K. & Beedle, M. (2002). Agile Software Development With Scrum. ISBN 978-0130676344
Seer, K. (2005) *The business rules awareness survey*, Business Rule Journal
Stapleton, J. (1997) Dynamic systems development method – the method in practice. ISBN 978-0201178890
Stokes, M. (ed.) (2001) Managing engineering knowledge, MOKA: methodology for knowledge based engineering applications, ISBN 9701860582950
Sure, Y. & Studer R.(2002) On-To-Knowledge methodology – final version. Technical report, Institute AIFB, University of Karlsruhe.
Weitzel, H.R. & Kerschberg, L. (1989) Developing knowledge-based systems: reorganizing the system development life cycle *Communications of the ACM* 1989:45-11, pp 16-19.

Yen, J & Lee, J. (1993) A task based methodology for specifying expert systems *IEEE Intelligent Systems* 1993:8-1, pp. 8-15
Zacharias, V. & Borgi, I. (2006) Exploiting usage data for the visualization of rule bases. *3rd International Semantic Web User Interaction Workshop*.