Goal-Driven Software Development

Ingo Schnabel
itestra GmbH
Ludwigstr. 35
Germany – 86916 Kaufering
schnabel@itestra.com

Markus Pizka
Institut für Informatik
Technische Universität München
Germany – 85748 Garching
pizka@in.tum.de

Abstract

Established software development processes focus on delivering software within time and budget according to a set of requirements. However, practical experiences show that neither business processes nor requirements can be fully understood in an early stage of a realistic software project. This is not primarily due to inadequate requirements elicitation but the fact that the technical implementation constitutes a formalization of a more or less fuzzy business domain revealing gaps and inconsistencies. Furthermore, the technology used adds constraints and enables new processes. Hence, trying to set the requirements in advance causes change requests, cost and time overruns, or even project cancellation. This paper continues the line of thought of iterative process models by regarding software development as a process iteratively converging business goals and technology from both sides. This "goal-driven process" is successfully applied in real-life commercial software projects and has repeatedly contributed to low cost but high quality software.

1 Convergence Vs. Refinement

The role of information technology for business processes still seems to be underestimated. Software is no longer only a means to an end but has itself an enabling role and conversely changes business processes. Over the last 30 years, the rapidly increasing complexity of software systems along with the growing economic impact of large scale software projects gave rise to disciplined software development processes, such as the waterfall model [20], the spiral model or the Rational Unified Process [12], more lately. However, the recent emergence of agile [5] approaches promoting quite different development strategies demonstrates that software development processes are still far from having reached a stable or even finished state and significant improvements are still possible and needed.

1.1 Are Requirements Overrated?

In this paper, we put one specific aspect of software development processes into question that is the prevalent strong concentration on requirements. As to our experience, the wide-spread misconception and overemphasis of requirements as an interface between business and software units repeatably causes excessive costs and reduced quality of the outcome. There are two main reasons for this:

1. Requirements are usually not identical with business objectives because the selection of requirements on the software part of the overall solution is usually already coined by the limited knowledge of the author of the requirements document about technical possibilities and their costs.
   E. g. for persons with non-technical backgrounds it is hard to understand that a certain behavior of a control on a graphical user interface might become excessively expensive whereas an extensive set of statistical analyzes might come almost for free on top of a relational database. Requirements documents written by these persons tend to include unnecessary expensive wishes while excluding technically simple features that would provide substantial benefit.

2. The development of a software system corresponds to a formalization of the supported business process. This formalization usually reveals inconsistencies and gaps within the current or target business process which in turn must be compensated with changes to the business process or the role of the software system.

The result of these two effects is usually a large number of change requests during and after development entailing time an cost overruns. We argue that this is one of the main reasons, why user involvement stands in first place of project success factors. Vice versa, lack of user input as well as incomplete or changing requirements are top of the list in project challenge factors in the CHAOS reports [1].

1.2 Goal-Driven Development

Instead of refining requirements down to an implementation we therefore recommend trying to find an optimal mapping between business objectives and capabilities of the technical platform in an iterative process. Figure 1 illustrates this subtle difference that has far reaching consequences on several roles and activities.

While established software processes (on the left) refine requirements down to an implementation (hopefully iteratively to reduce risks), the Goal-driven Development Process (right) suggested in this paper equally considers and adjusts business goals and technical aspects to come to an optimal solution corresponding to the current situation.

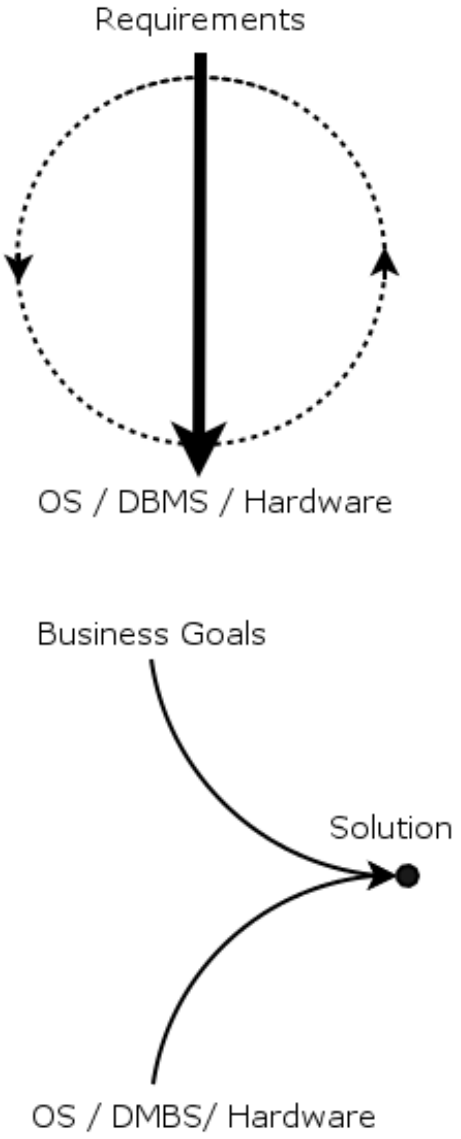

Figure 1. Top-Down versus Convergence

Outline The following section 2 explains the key principles of the goal-driven process proposed in this paper before we detail concrete activities of the goal-driven process in section 3. Section 4 discusses our practical experiences with the goal-driven process. This paper concludes with an overview on related work in section 5 and a brief summary in section 6.

2 Key Process Principles

Identifying goals before setting the requirements influences several core process principles. The following paragraphs describe how goals can be identified and how this affects top-down versus bottom-up orientation, team organization, roles and project size.

2.1 Collaborative Goal Identification

Our conception of business goals is closely related to the Goal-Question-Metric paradigm [3]. A top-level goal is defined as an informal description of what a stakeholder wants to change or improve in his business environment. Sample goals are: “increasing our knowledge about items on stock” or “increasing the cost efficiency of marketing events”. In contrast to this, invalid goals are “the system should provide a listing of all items in stock in foo bar format . . . ”. As shown in figure 2 every goal may have one ore more subgoals.

A set of questions is linked to every goal, which characterizes the way how software will be tested against defined goals after an iteration. A goal is attained, if every question of the goal is answered positively and all its subgoals are attained. Artifacts (such as requirement specifications, documents, source code etc.) can be assigned to leafs of the goal tree.

Stakeholders and contractors have to collaborate closely to work out goals so that the stakeholders know what is feasible and contracts gain a deep understanding of the business processes respectively. While goal definition is top-down driven, deciding, if a goal is feasible is bottom-up oriented. In other words, the collaborative identification of goals brings knowledge of users and software developers together.

2.2 Top-Down and Bottom-Up Convergence

Most established processes refine requirements top-down to the implementation. The main advantage of top-down orientation is, that it allows a horizontal team organization.

In contrast, bottom-up approaches try to provide generalized and therefore highly flexible and reusable compo-

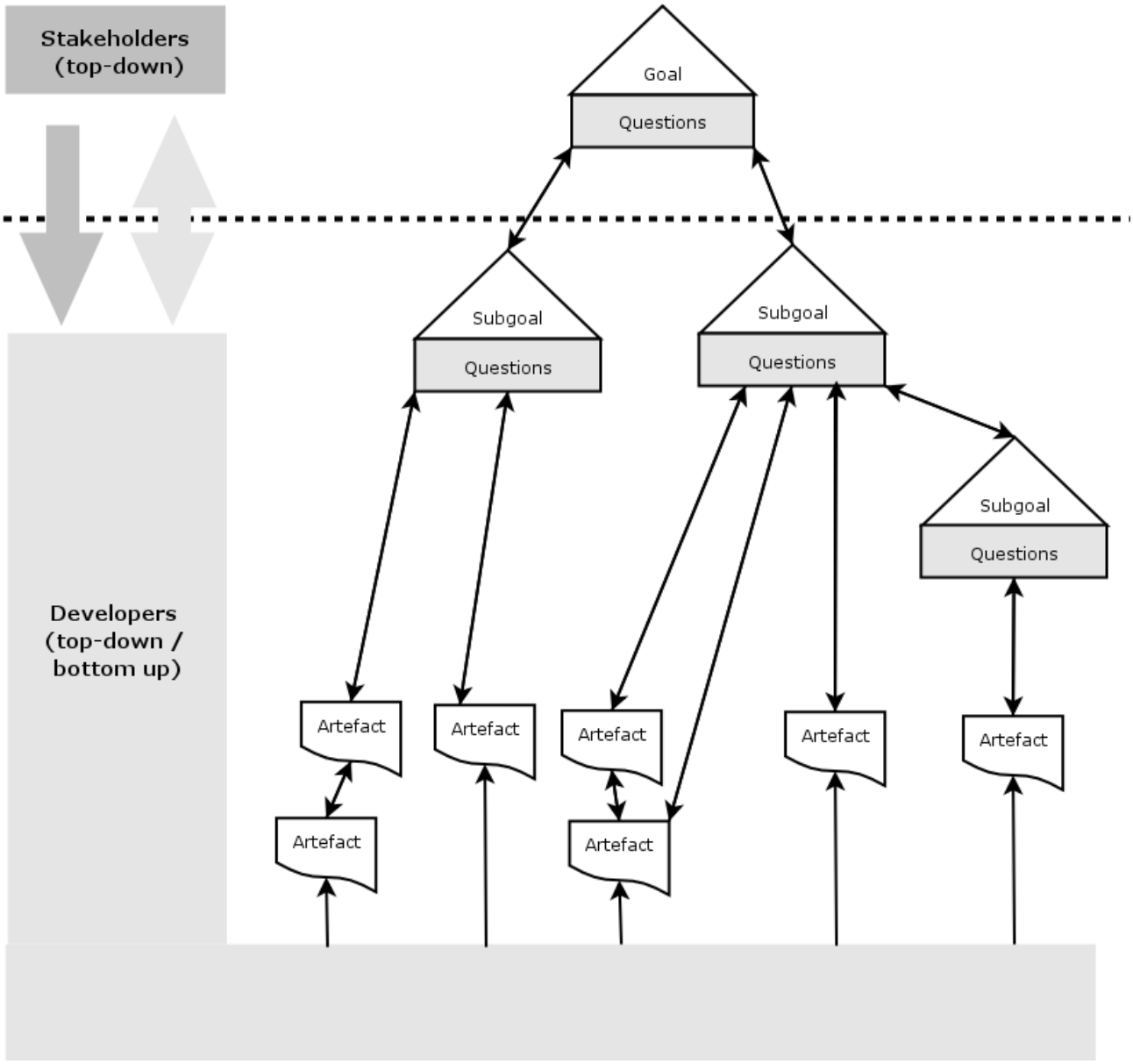

Figure 2. Goals and subgoals

nents or services. In an environment of constant change and evolution, the bottom-up design approach produces systems that provide superior satisfaction to users compared to top-down developed ones [14, 18].

In the GDP the collaborative identification of goals allows to combine top-down with bottom-up aspects.

We call this top-down thinking and bottom-up acting. As a result of a consequent top-down and bottom-up convergence a software developer is forced to maintain all artifacts appendant to a specific goal entailing a vertical team organization.

2.3 Vertical Team Organization

In contrast to horizontally organized project teams, where programmers implement the solution specified by the modeling team, the vertical organization implied by the GDP requires skilled and qualified generalists.

Every software developer has to be creative in order to find the best technical solutions corresponding to a high level business goal but is also responsible for delivering the implementation, because, again it is to expect that goals and subgoals need to be revised bottom-up according to knowledge gained during implementation.

Obviously, a horizontal organization is often chosen in real life software projects, since many organizations try to decrease overall development time by increasing the number of people working on a software project though this clearly disregards software engineering literature [11]. However, providing specific positions that only require a narrow qualification, such as full-time modelers, greatly facilitates over staffing software projects. As to our experience a slim vertical project organization is by far more productive than a horizontally organized project with numerous specialists and communication interfaces between them. Table 1 summarizes some important differences between a vertical and horizontal team organization.

The Unified Process is very clear about the fact that individual developers can and should take multiple roles on a project. Unfortunately, this advice still seems to fall on deaf ears within many organizations. In reality organizations tend to fill each role, such as requirements specifier, system analyst, user-interface designer, and database designer, with different people. The resulting communication overhead as well as emerging conflicts increase development time and costs dramatically.

2.4 Roles and People

Because of its vertical organization the GDP requires skilled generalists with the ability to fulfill many roles of the process. The GDP distinguishes the following key roles.

Programmers It is well known that the best programmers are up to a thousand times more productive than the worst, but the worst outnumber the best by a similar margin [6, 10]. Therefore, the programmers are the most important and therefore most expensive experts. They make the preliminary design and are responsible for top-down and bottom-up convergence. They have the main influence on delivering software in time and budget.

Business Analyst Business analysts are important to understand the stakeholders' business processes. The business analyst collaborates with the programmers during goal identification and later-on during testing. The business analyst should have a deep understanding of the stakeholders' business domain.

Software Architects The software architect's role [9] is closely connected to the programmers. In contrast to a programmer who focuses on one specific goal, the software architect keeps an eye on the whole project.

Project Manager The primary task of a project manager is to organize the project by assigning resources, keeping track of time and effort, and creating a productive environment for the project team.

Requirement Engineer To the extent of our experience, the role of requirement engineers and designers is often vastly overestimated. If once only job is to produce models, then there will be a strong tendency to over model things, as naturally everyone wants to do a good job.

People Once again, without any doubt the three most important ingredients to achieve efficiency and productivity in software projects are skilled, qualified, and motivated people. Surely, this insight is not new for itself. Early work on the importance of people for successful software projects dates back to de Marcos' Peopleware [8]. Agile processes such as XP and others [5, 4, 2] also focus on individual skills and communication. Practices such as pair programming aim at increasing the skills of individuals within the team, too. Collective code ownership and refactoring practices are further examples of how agile methods try to assign responsibility while requiring and increasing skills. As Grady Booch said, *People are more important than processes* [7]. The GDP follows this insight through vertical team organization and planning only a few different roles.

2.5 Minimizing Project Size

Another commonly known key to success in large project is to minimize project size in all aspects. Minimizing

Vertical Team Organization	Horizontal Team Organization
Few but highly qualified staff needed	Low qualified staff can be used
Low communication overhead	Staff is exchangeable
Everyone has to be familiar with (sub-)goals to which he is assigned	Many team members do not know any business objectives
Parallelization is simple to gain	Parallelization limited due to sequential order of activities
High motivation and strong identification with result	Lower motivation due to reduced responsibility for overall result

Table 1. Vertical versus horizontal team organization

project size does not only mean to limit the number of goals and software artifacts like documents, requirement specifications, models, etc. but also to limit the number of staff, to avoid mutual waiting (e.g requirements must be defined before implementation in established processes) and the size of the code. We know from experience that by consequently choosing a simple solution, the development time and the number of lines of code can both be reduced by about 50% compared to the application of flexible frameworks that might eventually become useful one time in the future.

Minimizing size is the simple most effective measure to increase maintainability and changeability of the system to new and changing business processes in the future. Note, that business processes are the most likely factor to change over the time [17]. Changes of the database system or the technical environment a rather rare and unpredictable so that highly sophisticated complicated frameworks can hardly be justified.

3 Goal-Driven Activities

Figure 3 depicts the core activities of the GDP and shows how their organization in iterations.

Every iteration starts with the identification of business goals and their priorities and ends with a running version of the software system corresponding to the selected goals. While incremental development of the software system is also done in other processes, we extend the scope of iteration to include a discussion of business objectives after each iteration, because we frequently experience that the business objectives themselves mature with the availability of usable implementation.

3.1 Goal Identification

The definition of goals is done in small groups of at most 5 people consisting of stakeholders and/or business analysts, and programmers. The stakeholders or business analyst have the responsibility to explain their goals to the programmers without taking any technical aspects of a possible solution into account. The responsibility of the programmer is threefold. First, they have to keep the business people' attention focused on business objectives instead of technology. Second, they have to offer ideas on what could be done with information technology to achieve these goals. Third, they have to inform the business side on the different costs of possible alternatives to allow prioritization of goals.

3.2 Vertical Distribution of Tasks

Based on the priorities of goals and individual skills of programmers, selected goals are assigned to groups of at most 4 programmers. Depending on the project one programmer may become associated with several goals vice versa. From there on, every programmer team decides for itself within a given boundary what (product) will be built how (activity), by whom and when.

3.3 Implementation and Testing

Implementation and testing are the most important process disciplines and receive the greatest amount of time because implementation and testing are the only activities that directly affect product quality. In simple words, if the software system is well implemented and corresponds to the business objectives, the project will succeed!

The GDP distinguishes between implementation-driven and goal-driven tests. Implementation-driven tests are done by the programmer permanently during implementation.

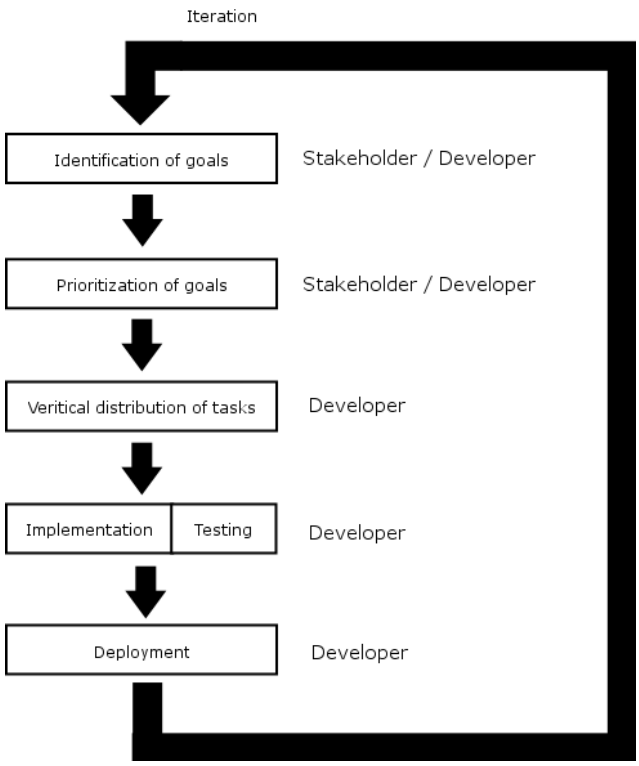

Figure 3. GDP iteration

This includes writing and executing test cases as well as manual testing. Goal-driven tests are executed at the end of each iteration and compare the actual implementation with the questions formulated during goal identification. Note, that a lack of accordance, might indicate a defect in the implementation as in conventional testing but it might also indicate a weakness in the definition of goals. Both possibilities are taken into consideration.

4 Practical Experience

The process sketched in this paper has already proven its benefits in commercial software projects[1]

One example is a large-scale project with a total of more than 50 man-years, critical interest to the stakeholders and a tough deadline. Here, the GDP contributed to a 30% cost reduction comparing the development costs of 2005 with 2004. After the introduction of the GDP, the schedule of the development effort was considered as reliable for the first time since this project was initiated.

Previous to the introduction of the GDP, about thirty people were involved and organized in a horizontal team structure; i. e. individuals were permanently allocated to single roles like software architect, designer, or developer. Many of the team members were hired from a highly regarded software service provider in Germany to increase the qualification within the project team. After one year of work and nearly about 1 million Euros were spent, hundreds of documents were written, but hardly any software implemented. As a consequence the project was canceled and because of its importance restarted with the GDP and a reduced team size of 8 programmers and 2 business analysts. Within five months 200000 lines of code were written, tested and about 100 change requests were successfully built into the system.

The stakeholders now feel, that the resulting software improves their business processes and are highly satisfied. As part of this success, the stakeholders have decided to proceed with the GDP.

5 Related Work

The work described in this paper has obviously a strong relationship with process models such as the waterfall model [20], the V-Model[2][16], or object-oriented process models like the rational unified process [13]. Statements such as the need for skilled personnel, minimizing project size and delivering early and often can also be found in agile

[1] Due to non-disclosure agreements, no details that would allow to draw conclusion about the site and purpose these projects can be published.

[2] Standard used in German government projects

methods [2] like Scrum [19] or XP [4].

However, there is at least one significant difference between the GDP described in this paper and other process models. Virtually all other processes (even agile methods) focus on requirements and try to map them top-down to an implementation. The GDP regards goals instead of requirements and combines them with technical capabilities, which leads to an integrated top-down and bottom-up approach that yields superior benefit for the user. Organizational changes, such as vertical team organization, are a consequence of this goal-orientation.

The idea to incorporate a bottom-up orientation into the development process is further influenced and supported by work on software evolution [15]. For example, the report about the successful long-term development and use of McIDAS [14] comes to the conclusions that a) "*requirements are not of paramount importance, user satisfaction is*" and b) that long-term success was not possible without a stubborn bottom-up approach. [18] gives further examples for the advantages of bottom-up orientation.

6 Summary

This paper introduced the basics of an iterative and incremental software development process, called the goal-driven development process (GDP). Clearly, the GDP is no revolution and has many similarities with other process models. However, the subtle differences of the GDP have various consequences and can make a big difference for the output of the process.

The two key contributions of the GDP are its concentration on business goals instead of requirements and the explicit integration of a bottom-up part. The combination of this two principles reflects the mutual dependency between business processes and information technology since information technology is not only a means to an end but it also enables new business processes. Practical experiences show that the GDP is capable of delivering high quality software at low cost.

As to our experience, the main challenge but also one of the main benefits coming along with the GDP is keeping the user of software focused on business aspects instead of technology. Currently, business people are still putting an amazingly strong focus on technical aspects in their requirements specifications, such as requiring the use of certain languages, frameworks, architectures and standard products. In fact, technology seems to be the only motivation for many projects, such as a migration to a web-based solution! What the GDP is trying to suggest is that business users would be much better off with staying focused on their business expertise and collaborating with skilled technical people for the implementation of their goals.

References

[1] The chaos report. Technical report, Standish Group, 1994.

[2] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta. Agile software development methods. Review and Analysis VTT Publication 478, VTT Electronics, 2002.

[3] V. R. Basili, G. Caldiera, and K. D. Rombach. Goal question metric paradigm. In J. J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 527–532. John Wiley & Sons, 1994.

[4] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 2000.

[5] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for agile software development. www, 2001. http://agilemanifesto.org/.

[6] B. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[7] G. Booch. *Object Solutions: Managing the Object-oriented Project.* Addison-Wesley, 1996.

[8] T. DeMarco and T. Lister. *Peopleware: productive projects and teams*. Dorset House, New York, 1987.

[9] M. Fowler. Who needs an architect? *IEEE Software*, 20(5):11–13, Sept. 2003.

[10] J. Greenfield. The case for software factories. *Microsoft Architects Journal*, (3), July 2004.

[11] F. P. B. jr. *The Mythical Man-Month*. Addison Wesley, 1995.

[12] P. Kruchten. *The Rational Unified Process: An Introduction, Second Edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[13] C. Larman, P. Kruchten, and K. Bittner. How to fail with the rational unified process: Seven steps to pain and suffering. Technical report, Valtech Technologies and Rational Software, 2001.

[14] M. A. Lazzara, J. Benson, R. Fox, D. Laitsch, J. Rueden, D. Santek, D. Wade, and J. T. Y. T. Whittaker. Man computer interactive data access system (mcidas): 25 years of interactive processing. *Bull. Amer. Meteor. Soc*, jan 1999.

[15] M. Lehman and J. F. Ramil. Rules and tools for software evolution planning and management. *Annals of Software Engineering*, 2001.

[16] M. Meisinger, A. Rausch, M. Deubler, M. Gnatz, U. Hammerschall, I. Küffer, and S. Vogel. Das V Modell 200x – ein modulares Vorgehensmodell. In R. Kneuper, R. Petrasch, and M. Wiemers, editors, *11. Workshop der Fachgruppe WI-VM der Gesellschaft für Informatik e.V. (GI) zur Akzeptanz von Vorgehensmodellen*. Shaker Verlag, 2004.

[17] T. Panas, W. Löwe, and U. Aßmann. Towards the unified recovery architecture for reverse engineering. In B. Al-Ani, H. R. Arabnia, and Y. Mun, editors, *Proc. of the Intern. Conf. on Software Engineering and Practice SERP'03*, volume 1, pages 854–860, Las Vegas, NV, June 2003. CSREA Press.

[18] M. Pizka and A. Bauer. A brief top-down and bottom-up philosophy on software evolution. In *Proc. of the Int. Workshop on Principles of Software Evolution (IWPSE)*, Kyoto, Japan, Sept. 2004. IEEE Computer Society.

[19] L. Rising and N. S. Janoff. The scrum software development process for small teams. *IEEE Softw.*, 17(4):26–32, 2000.

[20] W. W. Royce. Managing the development of large software systems. In *IEEE Wescon*, pages 1–9, 1970.