Insights into Decision Making

Analogies from Other Disciplines

Andreas Rüping
Sodenkamp 21 A, 22337 Hamburg, Germany
andreas.rueping@rueping.info
www.rueping.info

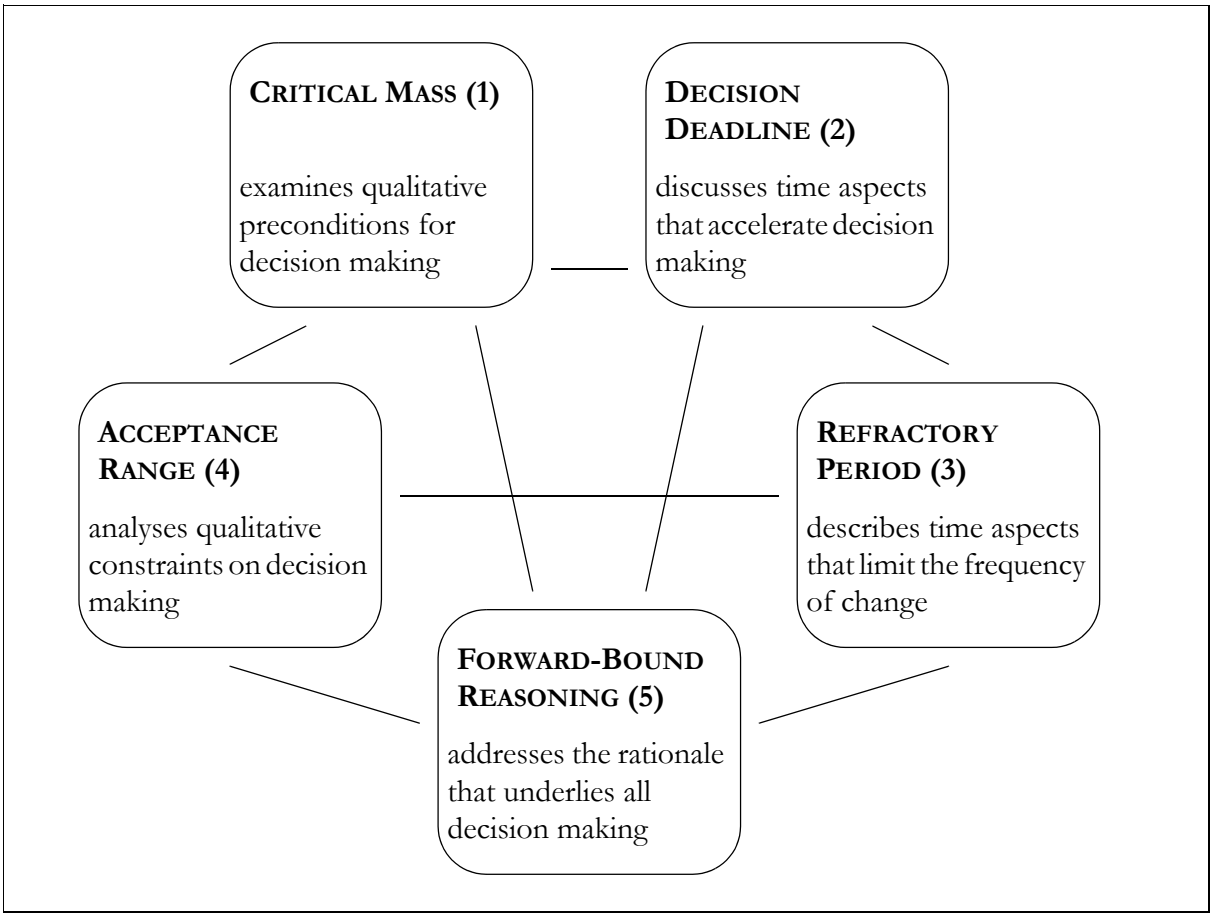
Introduction

Making decisions is part of our everyday life, and software projects are no exception. What architectural style would suit our application best? Should we go for the simpler design, or the more complex one that in return offers more efficiency? Should we use generic containers in our API or will typed objects be the better option? Who of the available team members should be working on the project? And who should be the team lead? Should we agree to the deadline the customer wants, or would it be safer (and more honest) to announce we'll need a month longer?

Questions like these force us to make decisions, and each such decision must aim at the best possible solution, whatever 'best' means in a specific context. Yet as we learn more and as conditions change, we may find that our decision wasn't perfect and that we have to adapt our strategy. As cognitive psychologists don't fail to remind us, questioning past decisions and looking for better strategies is an integral part of the human mind (Pinker 1997). However, changing decisions too frequently will cause ambiguous and inconsistent behaviour, and will result in a lack of decisiveness.

The goal must be to find strategies that are both adaptive and straightforward. This paper takes a look at other disciplines in search of suitable analogies: physics, biology, physiology, control engineering and economics.

The paper presents five patterns which all address a context that is best described as the decision-making processes in software engineering, and it takes its examples mainly from this area. The underlying principles, however, can be applied to decision making in general. The following diagram gives an overview.



1 Critical Mass

Problem **There's probably no limit to the information that could possibly influence a decision you have to make. How can you come up with a solid decision without turning information gathering into an endless process?**

Forces Whatever role you take on in a project, making decisions is part of your job. Your decisions may address questions as diverse as who'll be joining the team, what design is best, or what deadline you can agree to. In either case, you need some information on which you can base your decision. And obviously, the more information you have, the more you can use.

There is, however, an almost unlimited amount of information that you might take into account. There's always some additional information that might help you and that you might consider. If you collect all the information that may affect your decision, you'll be collecting information forever. This is not what you want.

Fortunately, this isn't what you need either. What you actually need is sufficient information — the amount of information that allows you to check the important criteria that your decision has to meet.

Analogy A nuclear fission occurs when a free neutron is captured by an unstable nucleus of a fissile material and the nucleus splits into two or more nuclei. A nuclear fission can produce free neutrons, which can then be captured by other nuclei. A nuclear chain reaction occurs when a nuclear fission produces, on average, at least one free neutron so that one fission can cause another. The critical mass of fissile material is the amount needed for a sustained chain reaction. The critical mass depends on the material, its shape and its purity.

Solution **Make yourself familiar with the idea that sufficient information, rather than complete information, can be the basis for a solid decision. Each decision requires a critical mass of information. Surplus information may be useful, but it isn't crucial for your decision.**

The critical mass depends greatly on whatever concrete decision you have to make. In particular, the size and the criticality of your project have an influence. It's therefore wise to determine the critical mass before gathering information. Here are some examples:

- When you have to make a design decision, a few experienced people should be confident that the proposed solution is technically sound. But there is no need to bring in more and more people — from some point onward, more opinions will probably hinder rather than help.
- When staffing a project, it's important to choose the right people. But that doesn't mean you have to check all details in their CVs. Talk to the people: if you get the impression they bring the right skills, things are probably fine.
- When you have to decide on a delivery schedule, avoid the pitfall of making plans that are too fine-grained. Breaking the project down into work packages that can be measured in weeks probably gives you the information you need.

If your decision is based on incomplete information, it's likely that there are some open issues left. You'll have to keep track of these open issues, so that you can address them when you revisit your decision after a REFRACTORY PERIOD.

Moreover, a decision based on incomplete information might be premature. This is a liability you have to accept in order to stay manoeuvrable. You can reduce the associated risk of failure by making series of smaller decisions rather than one big decision: an iterative and incremental process limits the effect of failure (Henney 2004).

Example A team developed the graphical user interface for an airline's check-in software. The customer had only vague ideas of what the user interface should be like, in terms of functionality and graphical appearance. There was no detailed specification. So what should the team do?

The team discussed several use cases with the customer to achieve a common understanding. And although these use cases were incomplete, the team felt that they represented the critical mass of information necessary to get started. The team chose to build a prototype that would roughly match the use cases. The lack of further details was no problem — any additional details would be subject to change anyway since the customer's representatives might change their minds the moment they would get to see the prototype.

Examples outside software

Other instances of this pattern can be found outside the daily project business, even well beyond software:

- Before taking on a new job, people need sufficient information on what they can expect and what people they'll be dealing with. But it's impossible to know everything beforehand. The critical mass of information is that which allows people to feel confident that they're making the right move.
- When people look for a house to live in, they often keep a checklist of criteria their new residence will have to meet. Such a checklist can do without many details, but it must contain the critical mass of information — those aspects that are crucial for a decision.

2 Decision Deadline

Problem

In an attempt to come up with the best possible solution, people are sometimes tempted to look for useful information endlessly. How can you fit decision-making into the schedule of your project?

Forces

There's no question that important decisions take time. You have to check various options, evaluate the pros and cons, and come up with a conclusion. You need the CRITICAL MASS of information before you can make a decision.

Some information, however, can be hard to find or can be virtually impossible to obtain. Or there can be so many information sources that your search for information could go on and on endlessly. Either way you're getting nowhere fast.

But you are probably bound to schedules in a project so you cannot delay your decision forever — at some point you have to decide. Others are waiting for your decision. If you fail to decide, someone else will probably make the decision, which wipes out the influence that otherwise you would have had.

Analogy

Many birds build their nests after the mating season. They search their habitat for the ideal place for a nest — a place that offers the best protection from predators, the best protection from the weather, and the best environment for raising their young. Yet the search for the best possible place is driven by a time limit: the nest has to be ready before the eggs are laid and incubated. By instinct, birds begin building their nests in due time.

Solution

Once you know that you have to make a certain decision, set yourself a deadline. Collect information until the deadline and then decide.

The time span that you give yourself should allow you to assemble the CRITICAL MASS of information that is necessary for your decision. Such time spans can therefore vary a lot:

- When you make a design decision, don't wait until you have evaluated all options to the finest detail. Get an overview of what options you have, ask others for feedback, and follow your gut feeling.

- Assembling a team requires that you choose the right people, but by the time a project starts, the decision must be made. Allow for enough time to talk to potential team members and to get to know them a bit.
- A customer expects a rough schedule before a project begins. That's only natural, and you have to come up at least with a rough estimate before you can analyse the job to the finest detail.

Working towards a decision deadline is essentially a time-boxing strategy that will help you focus on the criteria that are crucial for your decision. The advantages of time-boxing also materialize when the criteria keep changing, as is the case, for instance, when you are buying technology: there will always be new products or new versions, so if you don't set yourself a deadline, you'll postpone your decision forever.

If, once the deadline approaches, you feel that you still haven't reached the CRITICAL MASS of information, you have to decide whether you need to defer the decision or whether you'd rather make it anyway. This comes down to the question whether quality or schedule is more important — which can only be determined individually.

When you have made your decision, you cannot assume that it will hold forever. Conditions might change, or you weren't able to take all important criteria into account. Either way, it's important you revisit your decision regularly, ideally after a REFRACTORY PERIOD has passed.

Example

A team did a product evaluation of content management systems for a customer. The customer wanted to re-organise their web site soon, so the evaluation was driven by a deadline that was rather strict: the team had no more than two months for the requirements analysis and the actual product evaluation. Naturally, the team organised their job within the time frame that was given to them. They interviewed the customer to find out what the key requirements were. They weren't able to analyse all details of the workflow processes the customer had in mind, but they were able to identify a small number of crucial criteria.

The team used the time they had for a brief market study of content management systems. They considered current releases only. Obviously future releases would include improvements and extensions, but the team understood that if they waited for the perfect product that matched the requirements best, they would wait forever. The team established a short-list of only three vendors based on the crucial criteria. The team then ran workshops with these vendors which provided insight into the three systems on the short-list. The team made a recommendation based on this information.

Examples outside software

You come across decision deadlines outside software as well:

- A job decision is one that cannot be delayed forever. Someone who is offered a job is normally expected to make a decision within a few weeks.
- Looking for a perfect house can last for ages. There's always the possibility of finding an even better house one day. But if people argue along these lines as an excuse to postpone their decision, they'll never move in anywhere.

3 Refractory Period

Problem **Immediately after a decision is made, there is a tendency to question the reasons and to revise the decision. How can you prevent decision-making from becoming an unstable process?**

Forces Even if you have made the best possible decision, you must be aware that the basis for your decision can change: customer requirements can change, the programmers might gain new insight into the trade-offs between design alternatives, or team members might become unavailable. You'll have to check whether your decision turns out well and adjust your strategy if necessary. The agile software development methods advise us to reflect regularly on what we do and to check whether our strategy needs to be adapted (Cockburn 2001).

However, if you correct your decisions and change your strategy too frequently, your project will make only little progress. People will not trust you if you change your strategy too quickly and so suffer from a lack of decisiveness.

Analogy A heart contracts when it receives an impulse over the nervous system. Directly after the heart has received this impulse a period sets in during which the heart will not react to any further impulses and therefore will not contract again. This is the so-called refractory period (which for the human heart lasts for about 0.1 seconds). Evolution has provided the heart with a mechanism that protects it against impulses that occur too frequently and that wouldn't allow a coordinated contraction to be carried out.

Solution **Once you have made a decision, define a time span that you allow to pass before you reconsider your strategy. After that time span has passed, correct your decision if necessary.**

What that time span is depends greatly on the concrete decision — you'll have to define it individually. It should be the minimum time span that you'll need to try out the option you have chosen and to gather enough information to re-evaluate your decision. The following scenarios give a good estimate:

- After making a design decision, allow at least a few weeks to pass before re-evaluating the design and use this time to see how the pros and cons turn out. If you choose to build a prototype (Coplien Harrison 2004), that time span might well serve as a good refractory period.
- After making a staffing decision, give the people involved a few weeks to familiarise themselves with the project before considering a change in the team.
- After agreeing on a delivery schedule, wait at least half the time until the next milestone before considering to revise the schedule.

During the refractory period, reflect regularly on what you are doing and collect information that suggests that your decision might have been wrong, or less then optimum, but don't take action yet.

Once the refractory period has passed, see to it that you come to a conclusion whether you'll need to adapt your strategy or not, and take action if necessary. Whatever the conclusion will be, it should be based on FORWARD-BOUND REASONING. If you decide to take action, the next refractory period sets in.

Example

A team had been discussing the software architecture for an e-government internet portal for a while. The main question was whether a portal server should be used or not. In the early stage of the project the team discussed the pros and cons of using a portal server, and as more people joined the team, more pros and cons were revealed. At some point the team felt they had gathered enough information and decided they would build a prototype without using a portal server.

As far as the prototype was concerned, the team wouldn't consider a change in direction. When the customer expressed doubts concerning the soundness of the architecture, the team made it clear that only the prototype would show. Once the prototype was completed, the team re-evaluated their design. They kept their original decision, though if there had been evidence that the alternative route would have been better, they would, at that point, have revised their decision.

Examples outside software

There are refractory periods in other decision-making processes as well:

- After taking on a new job, it is common practise to stay with this job for at least two or three years. This way people make sure they can contribute something useful to an organisation and avoid the impression of job-hopping.
- After relocating to a new place, most people feel the deep desire to 'stay there for a while' before they even think of moving again.

4 Acceptance Range

Problem

Changes to a system can make the system better. Yet when changes offer only small benefits, these benefits can be outweighed by the effort that is necessary to perform the changes. How can you prevent changes from doing more harm than good?

Forces

In the decisions you make in your projects, you always strive for the best possible solution. This should be the case whether you're making technical, management or team decisions.

However, it's only natural that after a while new ideas emerge that may let your decision look less than optimum: designs can be improved and management strategies can be refined. It is wise to look out for possible improvements — in fact the search for improvements is at the core of what is often referred to as a 'learning organisation'. As a consequence, revising past decisions is, in general, a perfectly natural process.

Still, there is no way to deny that revising past decisions can represent an overhead effort that is not to be ignored. Implementing a new design, training new team members, negotiating a different delivery schedule with the customer are all things that use up time and effort, and it's not always clear that this effort is justified by the expected benefits.

Moreover, all measurements undergo some uncertainty, so what seems to be a small improvement might in fact be no improvement at all. Revising past decisions for a tiny little improvement is therefore often counter-productive.

Analogy

In control engineering, temperature control is a common task. It is typically accomplished by defining a range of acceptable temperatures. For instance, if you want the room temperature to be 20° C, the temperature controller may assume a range of ±2° C: the heating is switched on once the temperature drops below 18° C, and is switched off once 22° C has been reached. This way, the heating system is prevented from switching on and off more or less continually and is immune to small measurement errors.

Solution

Once you have decided for one out of several options, revise your decision only if it turns out that another option is significantly better than your original choice.

But what is a significant improvement? By analogy, you should define an acceptance range around what you currently consider the optimum solution and take action only if the current solution lies outside that acceptance range.

The acceptance range implies a certain 'improvement threshold', the border beyond which an improvement becomes significant. Choose a better solution only if the potential for improvement lies beyond that threshold, as the following scenarios explain:

- Choose a better design only if that design is a significant improvement over the one you have now, and if that improvement isn't outweighed by the effort you'll have to spend on migrating the software.
- Replace a team member only if that person could do a much better job on a different project, or if there's someone else who could work clearly more successfully on your project. A small plus in skills or experiences is easily outweighed by the negative consequences of discontinuity and the additional training efforts for a new team member.
- Work out a new schedule only if it has clear advantages that are worth the reorganisation.

Once you have implemented a new solution, it becomes the standard against which you'll have to measure possible future improvements.

Example

A team worked on an internet portal of an insurance company. This portal was intended to be used by bank accountants for selling insurance products at the bank counter. Recently, a series of small changes had been made to the system. There had been changes to the layout, both by modifying the CSS styles and by changing the templates within the content management system. Also, the team had made several changes to the functionality that the customer had classified as

low priority. And although these changes were small, effort had to be spent on development, tests and deployment. The team felt they were more or less permanently in the middle of the delivery stage, despite the fact that the actual improvements to the system were only marginal.

At that point the team decided to perform no more miniature improvements. Refactorings would of course still take place regularly, whenever the expected benefits were large enough to justify the effort. For instance, one refactoring saw the extraction of common functionality to avoid redundant code. Another refactoring addressed the deployment processes and made them much smoother. These refactorings were significant and well worth the effort.

Examples outside software

The idea of this pattern — the principle that only significant improvements justify to take action — can also be found outside software engineering:

- A new job sounds like a good idea if that job offers more interesting tasks, more responsibilities, more money — or whatever someone may desire. And yes, it is a good idea to seize an opportunity. If, however, the new job offers only marginal improvements over the current job, it's probably not worth the risks associated with a job change.
- People hesitate to move into a new place, even if that place has advantages over where they live now. Unless the living conditions improve clearly, the troubles associated with relocating don't seem to be justified.

5 Forward-Bound Reasoning

Problem

People are sometimes reluctant to make a change in their direction since this can let a previous decision look like a mistake. How can you make sure you take the best possible direction?

Forces

Re-evaluation is an integral part of any sustainable strategy. Of course, there are good reasons not to permanently question decisions after you have made them, but to define a REFRACTORY PERIOD instead. But once this period is over, you have to revisit the decision and see whether it still holds, taking into account new information you have gained on the way. The re-evaluation may or may not suggest a change in direction.

Sometimes, however, people are scared to make a change in their direction, because such a change can be interpreted as admitting they made a mistake in the first place. People are then inclined to stick to their old decision, although that decision wasn't good, afraid of admitting a mistake.

But this is not a sound way of thinking. It is inevitable that people make mistakes. In a way it's natural that a past decision may need adjustment. The reasonable way to deal with mistakes is to learn from them and to make better decisions in the future.

Analogy When a stock goes down at the stock exchange, some people fall into the trap of trying to get their original investment back. They hang onto the stock, because they don't want to admit that their decision to buy was perhaps wrong. But this isn't a good strategy. Regarding buying or selling, all that matters is the future potential: if there are signs that the stock will go up again, people should keep it, but if it is likely that it goes further down, they should sell the stock.

Solution **When you consider a change in direction, only look forward. Base your decision only on the future potential and the costs of the various options you may have.**

There is no point in maintaining a decision of the past if that decision has turned out wrong. Your decision now shouldn't be driven by the arguments of the past.

This of course does not mean that you shouldn't look at the past and learn from it. It's wise to reflect upon what worked well and what didn't and to consider the conclusions for the future:

- You have to re-evaluate a design after a REFRACTORY PERIOD, for instance after you have used the design to build a prototype (Coplien Harrison 2004). This evaluation must be forward-bound, using techniques such as feedback meetings (Cockburn 2001, Schwaber Beedle 2002) and project retrospectives (Kerth 2001) to see whether a better solution can now be devised. If the old design lies outside the ACCEPTANCE RANGE of the better option, don't cling to the old design but choose the better option instead.
- Sometimes you have to admit that someone on a team isn't doing a good job. This isn't a conclusion you should jump to quickly, yet the situation may occur. If so, you have to look for the best solution for the future. Of course you have look into the reasons and to take possible hard feelings into account. But these are considerations of the future.
- Adjusting the schedules in a project can be hard. Still, if it becomes clear that you cannot meet a certain deadline, it's better to take action and to figure out a plan that will work fine in the future. Though such a move may be difficult to explain, it is still clearly superior to continuing on the grounds of a plan that you know will fail.

Learning lessons from the past is easier if you choose to document the rationale behind the decisions you make. It is the rationale, not the mere decisions, that can be useful once you have to revisit your decision and examine whether it is still appropriate as far as future potentials are concerned (Rüping 2003).

Example A team developed a database access layer which could also manage versioning. Requirement analysis revealed that two different algorithms could be used to store and retrieve versions of the data: one that used more storage and had slower write access, but offered really quick read access, and another that used less space on the database, but had the drawback that read access was somewhat slower. After a thorough analysis, the team assumed that read access had the highest priority and decided to implement the first option.

Almost two years later, the amount of data stored in the database had grown to such an extent that nightly batches took too much time. Although the amount of data (and the problems associated with it) had been difficult to foresee two years before, it wasn't easy for the team to admit the time had come to switch to the other implementation option. But there was no point in clinging to a solution that didn't work properly any more, so the second option was implemented.

Examples outside software

Forward-bound reasoning plays an important role in life in general. Again, this doesn't mean that we shouldn't learn from the past — but it's the future prospects that should drive our decisions:

- When people consider a career move, what matters is what they intend to do in the future and what options they have. There is no point in keeping a job if there are clearly better options for the future. Of course people may have the wish to complete a job or to stay loyal to an organisation, but that is still looking forward.
- When people are going to move and look for a new place, it's a wise strategy not to look for an exact copy of the place they had before, but for a place that suits their needs best. The decision must be driven by how people would like to live, not by the features the old place had.

Conclusions

As you will have noticed, some of the patterns in this paper aim to accelerate the decision-making process, while others reduce the frequency of change. As a whole, the collection of patterns is intended to demonstrate the tension that exists between the need to move forward and the need to establish a stable process.

At first sight, the patterns with an emphasis on steadiness seem to be opposed to the principles fostered by agile methods. Agile methods welcome change and express a need for reflection and the adjustment of one's behaviour (Cockburn 2001, Schwaber Beedle 2002).

But actually, this isn't a contradiction to what the patterns in this paper explain. The patterns do acknowledge the need for reflection and change. They do, however, discuss a limit to the frequency of re-evaluation and change, so that the overall decision-making process doesn't become unstable.

This is what we can learn from the analogies from nature: change can be too slow or too fast, and it is important to employ mechanisms that allow us to enjoy the balance.

Acknowledgements

First of all I'd like to thank Neil Harrison for the excellent shepherding for EuroPLoP 2004. The paper originally consisted of two patterns (REFRACTORY PERIOD and ACCEPTANCE RANGE). Neil offered valuable feedback on those and suggested I add more patterns. He came up with the idea for FORWARD-BOUND REASONING and contributed the analogy to DECISION DEADLINE.

Thanks to Götz Hemicker for an insightful discussion of the physiology of the human heart which refreshed my understanding of the refractory period.

Finally, I'd like to thank the participants of the EuroPLoP 2004 workshop in which this paper was discussed. The workshop spawned much encouraging and constructive feedback. Special thanks to Kevlin Henney for a long list of detailed comments.

References

Cockburn 2001

Alistair Cockburn. *Agile Software Development*. Addison-Wesley, 2001.

Coplien Harrison 2004

James O. Coplien, Neil B. Harrison. *Organizational Patterns of Agile Software Development*. Prentice Hall, 2004.

Henney 2004

Kevlin Henney. "Stable Intermediate Forms", in Klaus Marquardt, Dietmar Schütz (eds.), *Proceedings of the 9th European Conference on Pattern Languages of Programs, 2004*. Universitätsverlag Konstanz (UVK).

Kerth 2001

Norman L. Kerth. *Project Retrospectives: A Handbook for Team Reviews*. Dorset House, 2001.

Pinker 1997

Steven Pinker. *How the Mind Works*. Allen Lane, The Penguin Press, 1997.

Rüping 2003

Andreas Rüping. *Agile Documentation — A Pattern Guide to Producing Lightweight Documents for Software Projects*. John Wiley & Sons, 2003.

Schwaber Beedle 2002

Ken Schwaber, Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2002.