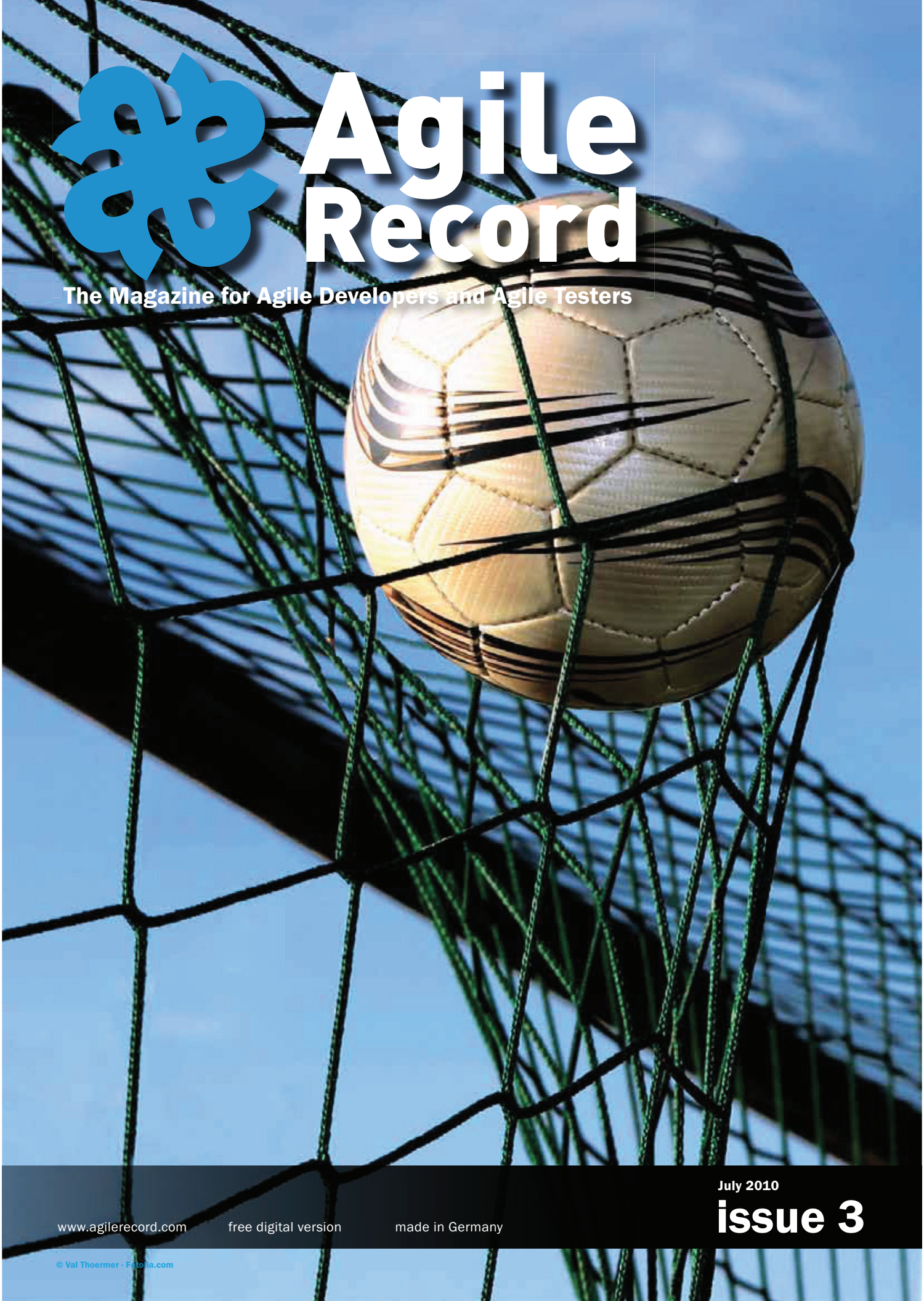

Agile Record

The Magazine for Agile Developers and Agile Testers

July 2010
issue 3

www.agilerecord.com free digital version made in Germany

Keep it simple – Seven rules of thumb for effective software development

by Remi-Armand Collaris & Eef Dekker

Introduction

An important question with respect to implementing a software development method is how to *concretize* the method in the organization and the project which it should serve. This step is easily forgotten in introducing and applying a new method, and it is easy to slip into the habit of regarding a full and clean application of a method as a goal in itself. Many development methods (DSDM, Scrum, XP) presuppose this concretization step without making it explicit. In the Rational Unified Process (RUP), this step is explicit in its first key principle: "Adapt the Process". It is by no means an easy step, since knowledge of and experience with the method is needed.

Which rules could you use in performing this step? How do you get to a clear and concrete development process that optimally supports your project? Which tools do you apply and who will use them? Below we have listed a few situations, in which this step should have been taken more consciously:

1. An enterprise bought an expensive tool but did not anticipate the large amounts of time it costs to implement it and keep it up-to-date. This results in the tool remaining unused.
2. Review forms are developed, but it takes so much time to fill them in that everyone avoids reviewing.
3. RUP defines 128 work products. It is decided that all need to be created. This results in the project coming to a complete standstill.
4. The project needs to be finished quickly, so a large development team is rolled out, but the demanding organization cannot supply sufficient input to the team.
5. The change procedure takes 3 months at least. This leads to a large part of the changes not being relevant anymore by the time they are finally implemented.
6. People responsible for requirements are not available during system implementation, and therefore the system does not comply with the customer's needs.
7. A lot of time was invested in keeping an up-to-date traceability between requirements and code, but estimation of changes does not appear to be any quicker.

In this article, we provide seven rules of thumb which help you to make the development method of your choice more effective. These rules were developed during our experiences in various enterprises, in which we helped to implement modern development methods. Following these rules will help to avoid the situations described above.

Seven Rules of Thumb

All rules presented here follow the same principle: balance cost and revenues. A measure (document, procedure, application or a tool) is effective if it yields the correct effect, i.e. an optimal balance of costs and revenues. Costs and revenues not only apply to the period of software development, but to the complete lifecycle of the application and to the context within which the application is developed and used. An important question in this respect is what the expected lifetime of the application is and how many changes are expected.

We often see people being so enthusiastic about the expected revenues that they lose sight of the costs. Or the other way around: because of the costs, a "smart" solution is not chosen, although in the long run the revenues would have outweighed the costs. Costs or revenues in themselves are not the criterion for a good decision, but the balance of both is.

Rule 1: Keep the level of ceremony as low as possible

The level of ceremony can be defined as the amount of additions to the development process for the sake of controlling the process and its resulting work products. A higher level of ceremony implies more formal recording, review and approval and often more work products, procedure descriptions and more detailed means of control.

Every organization knows its own “ceremonies”. Usually the bigger the organization, the more formal it is. Some examples of lower and higher levels of ceremony are listed in the table below:

Lower level of ceremony	Higher level of ceremony
Information is available in a WIKI, everyone can look there.	It is explicitly recorded who must be informed, and in what way.
The work products themselves mirror what is agreed upon.	Minutes of every meeting are made and formally approved.
Informal exchange of information	Information is usually exchanged in writing.
Quality is acknowledged and assured by the team.	There is a Quality Assurance department.
Changes can be processed easily.	There is a committee which must approve changes.
Reviews are held informally.	All review results are recorded.
Standards and guidelines are implicit.	There are extensive lists of standards and guidelines.

Many organizations are unaware of the costs as well as the revenues of their level of ceremony. Many people see formal reviews, an extensive approval procedure and detailed plans with Gantt charts as simply indispensable. We have found that asking for explicit estimates of costs and revenues helps people to become aware of the amount of overhead brought about by the level of ceremony.

Rule 2: Keep the team as small as possible

A development team of exactly 1 experienced person who unifies all roles in himself is most efficient in avoiding handovers. However, not only the availability of one person with all knowledge and skills needed prohibits this option, but also development speed (time to market), vulnerability (what happens if this single person gets sick) and safeguarding of knowledge.

Adjust the number of team members to the level of input the demand organization can supply on a continuous basis. It does not make much sense to install a team of 3 analysts and 8 developers if the maximum input the key users can supply is only sufficient for 1 analyst and 2 developers.

Moreover, a bigger team does not have a proportionally bigger productivity, due to a larger overhead in planning, handovers and dependencies. It may well be the case that a well-functioning team of 3 developers, an architect, an analyst and a tester are more productive than the same team but now enlarged with 3 more developers, an analyst and a tester to meet the time-to-market. If such a scaling operation takes place toward the end of the project to meet the deadline, the net effect will be negative. Many people forget to take into account the learning curve of new team members and the extra overhead.[1]

Rule 3: Make as few work products as possible

If you make a list of work products, the questions you need to ask yourself are:

- Who will miss this work product?
- What will go wrong if we don’t make it?

This approach will clarify the purpose of the work product. Will this work product only be needed for development, or will it also play a role in future system administration? By knowing who will need the work product, you can involve this person in producing it, validating its contents and balancing production effort with the expected revenues.

The amount of work products for software development depends on the circumstances of the project, as do their specific form and comprehensiveness. If a sketch on a whiteboard suffices, it is not necessary to produce a comprehensive document.

Some work products will be part of the final product, for instance because they are needed for system administration or future maintenance and support. If the only thing you deliver is the application, and the demand organization can use and administrate it, that is fine for the short term. This is even more likely if an Agile way of development is applied and all code is accompanied by unit tests and code documentation and kept as simple as possible by refactoring. However, most applications must be maintained long after the original developers are gone. In this case it might be better to have some documentation in place, like use case specifications, a software architecture document and system administration document.

Rule 4: Apply the simplest possible approval procedure on as few work products as possible

Not all work products need to be approved in a formal way. A similar question as above applies here:

- What goes wrong if a work product is not formally approved?

If the answer is “nothing”, it is clear that formal approval is not needed. Often the work products that define the scope of the project are candidates for formal approval. Also use case specifications could be approved in order to have a common point of reference for accepting the solution.

Often a few separate, specialized tools better meet the requirements than a complete integrated toolset.

Formal approval takes time and energy. On the other hand, it delivers clarity, baselines and reference points. If there is a clear mandate of one person approving a work product, this will simplify the process tremendously. For example, a subject matter expert who has mandate can decide very quickly if a use case specification is correct. He can do the formal acceptance as well. It does not add much value to have a person higher in the management hierarchy accept the use case specification.

Rule 5: Take the simplest toolset that meets your needs

It is often underestimated how well requirements and management can be done using only a big whiteboard, brown paper and post-its, a word processor, spreadsheet, simple modeling tool and WIKI. Add more only if there is a very clear need or bottleneck. Be sure that a solution is already proven in practice and that you have a good picture of how it will help you before choosing it. Do not only formulate requirements for a tool, but look at the costs as well. Tools need to be installed, configured and maintained. The time and expertise needed to do this is often underestimated.

Formulate tool requirements with concrete improvements in mind, which can be expressed in terms of revenue. If you balance costs (purchase, configuration, maintenance, keeping information up-to-date) against revenues (savings in time), this balance may easily turn out negative.

If you have a set of prioritized tool requirements, look at the simplest tools that meet them. Often a few separate, specialized tools better meet the requirements than a complete integrated toolset. Integrated toolsets are inherently complex and may lack the features of specialized tools.

Rule 6: Maintain (explicit) traceability as little as possible

Traceability may mean two things: there is a connection made between decisions in time, or between work products belonging together in one baseline. The connection may be made explicitly (in a tool or spreadsheet) or implicitly by following design rules or naming conventions. Traceability is often desired because it may help with:

- Keeping business requirements, high-level and detailed software requirements, test cases and implemented functionality in line;
- Impact estimation, especially when the original developers are no longer available;
- Tracing why certain decisions were made, by whom, and when.

Explicit traceability from high-level to detail software requirements and implemented functionality may be useful, for example in situations in which changed laws or government rules may lead to adaptations in the application. If the right traceability is in place, this may help in estimating the impact of these changes. Keep in mind, however, that traceability can never serve as a substitute for intelligent, human inspection.

Traceability may be used to ensure that all high-level requirements are met. While detailing requirements, you gain more insight. High-level requirements formulated (much) earlier might no longer be relevant. Hence, we need human inspection here as well. Maintaining traceability between changing requirements is time-consuming, even when tools are used. A solution using just human inspection might be cheaper.

Traceability can be kept implicit in many cases, which is a cheap approach. For example, naming conventions and object-oriented development may help. If something is called ‘dossier’ by the business, then it is a good idea to give the object in the code which represents the dossier the name ‘dossier’ as well. Things you can do in reality with a dossier will then be associated with the dossier in the code. The more perspicuous an application is built, the better the implicit traceability is.

The worst scenario is that in which explicit traceability is installed but not kept up-to-date. Outdated information renders the traceability unreliable and hence useless. In this case the investment in traceability has a negative result, since the effort that has been put in does not pay at all. Consider explicit traceability only when its maintenance is guaranteed throughout the application’s lifecycle.

Rule 7: Choose the most effective form of communication

The quicker and better your intentions are understood, the more effective your communication is. The best way to reach this is to have a live conversation and be enabled to visualize your intentions. The worst way is to send a text document without any explanation or possibility to raise questions about it. In the figure below, various communication channels and their effectiveness are shown.

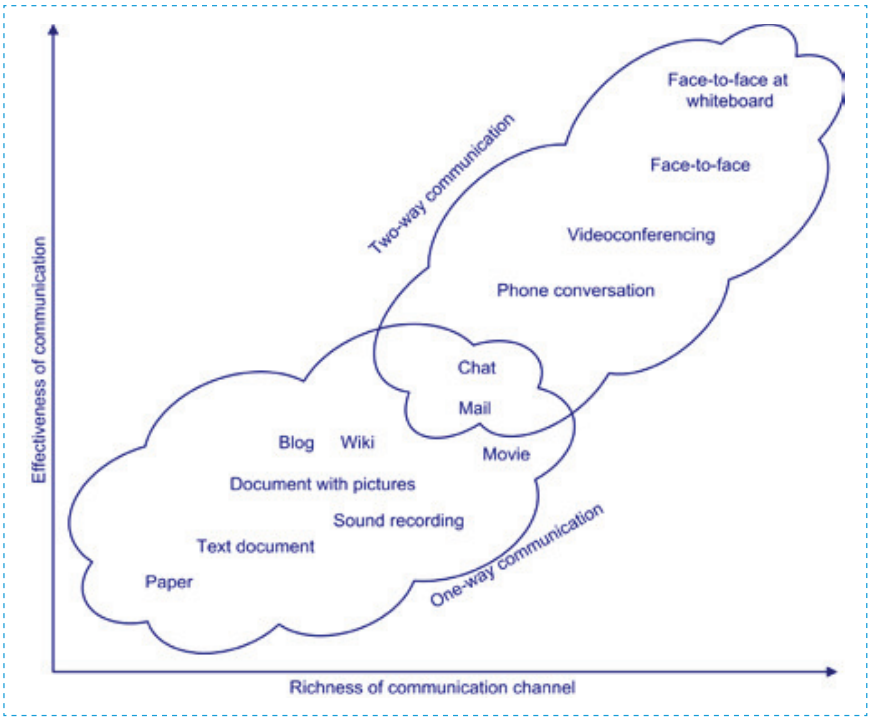

Figure 1: Effectiveness of communication channels

Do not trust blindly in this figure. Keep in mind how a communication channel is used. In order to convey an intention, a piece of paper on its own is not very effective. If we have a face-to-face conversation, a paper document could, however, be helpful to support the conversation and capture the result for future reference. It then functions as a means to support the "real" communication.

Conclusion

We have presented seven tested rules of thumb, which can be used to tailor any software development method. They help you to implement an effective development process or to improve an existing process:

1. Keep the level of ceremony as low as possible
2. Keep the team as small as possible
3. Make as few work products as possible
4. Apply the simplest possible approval procedure on as few work products as possible
5. Take the simplest toolset that meets your needs
6. Maintain (explicit) traceability as little as possible
7. Choose the most effective form of communication

Each of these rules supports balancing of costs and revenues. Especially revenues that will show up only in the long run might be hard to quantify, but might be very worthwhile investigating. Making expected revenues explicit gives you a tool for measuring the effectiveness of the measures taken. The results of such a measurement are in turn indispensable for further optimization of your development process. ■

[1] See Frederick P. Brooks Jr., *The Mythical Man-Month. Essays on Software Engineering*, Anniversary Edition 1995. The insights that Brooks formulated in 1975 are still valid.

> About the authors

Remi-Armand Collaris

Remi-Armand Collaris is a consultant at Ordina, based in The Netherlands. He has worked for a number of financial, insurance and semi-government institutions. In recent years, his focus shifted from project management to coaching organizations in adopting Agile using RUP and Scrum. An important part of his work at Ordina is contributing to the company's Agile RUP development case and giving presentations and workshops on RUP, Agile and project management. With co-author Eef Dekker, he wrote the Dutch book RUP op Maat: Een praktische handleiding voor IT-projecten*, (translated as* RUP Tailored: A Practical Guide to IT Projects*), second revised edition published in 2008 (see www.rupopmaat.nl). They are now working on a new book: ScrumUP, Agile Software Development with Scrum and RUP (see www.scrumup.eu).*

Eef Dekker

is a consultant at Ordina, based in The Netherlands. He mainly coaches organizations in implementing RUP in an agile way. Furthermore he gives presentations and workshops on RUP, Use Case Modeling and software estimation with Use Case Points. With co-author Remi-Armand Collaris, he wrote the Dutch book RUP op Maat, Een praktische handleiding voor IT-projecten*, (translated as* RUP Tailored, A Practical Guide to IT Projects*), second revised edition published in 2008 (see www.rupopmaat.nl). They are now working on a new book: ScrumUP, Agile Software Development with Scrum and RUP (see www.scrumup.eu).*