

Being Agile in a Non-Agile World

By David West, Managing Director Americas

Table of Contents

Introduction	2
The Agile Manifesto	2
The Essence of the Principles	4
SCRUM	5
Issues with Agility.....	5
The Three Perspective of Process	6
Funding and Organizational Lifecycle	7
Aspects	7
Mechanisms	7
Example of a lifecycle	7
Project Lifecycle / Team Approach	8
Aspects	8
Mechanisms	9
Example of a team approach	9
People Doing Work – Practice Approach	10
Aspect	10
Mechanisms	10
Example of an Engineering Practice	11
Summary.....	12
About the Author.....	12
About Ivar Jacobson International.....	13

Introduction

Agility is the topic of the day with many organizations disenchanted with traditional software development processes in favor of processes that focus on delivery and team dynamics. But these organizations are finding the adoption of small, agile approaches for development often hard to implement into organizations that are neither small, nor agile in nature. Organizations are finding the promise of agility is often compromised in response for the need for prediction, planning, and financial controls. Project managers are trying to balance the needs of the organization versus enabling teams to deliver software faster and of a higher quality. In this paper we discuss how to implement agile thinking in the context of both defined engineering practices and organizations Software Development Lifecycle.

The Agile Manifesto

The Agile Manifesto describes four main tenets that are found in teams that are highly effective and ‘agile’ in nature. These four tenets are;

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation

- Responding to change over following a plan

Agile Manifesto <http://agilemanifesto.org/>

The focus on working software, the team, the relationship with the customer and the ability to manager change are apparent 12 principles that were defined in support of the manifesto.

1. Our highest priority is to **satisfy the customer** through early and continuous delivery of valuable software
2. Welcome **changing requirements**, even late in the development. Agile processes harness change for the customer's competitive advantage.
3. **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must **work together** daily throughout the project.
5. Build projects around **motivated individuals**. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is **face-to-face** conversation.
7. **Working software** is the primary measure of **progress**.
8. Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to **technical excellence** and good design enhances agility.
10. **Simplicity**--the art of maximizing the amount of work not done--is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the **team reflects** on how to become **more effective**, then tunes and adjusts its behavior accordingly.

Agile Principles <http://agilemanifesto.org/principles.html>

What is interesting about the 12 principles is that these principles imply to some very tangible characteristics for a development team, its process and the infrastructure that supports it.

Principle	Implication
1. Satisfy the customer	The need to introduce a process that regularly delivers software that can be reviewed by the customer. The development is planned around the customer's priorities.
2. Changing requirements	Requirements cannot be signed off early in the process. This means that planning, development and testing must support a continuous changing set of requirements.
3. Deliver working software frequently	Need to build software early with development resources and environments being available from the start of the project.
4. Work together	The business needs to work very closely with the development team providing responses to questions and evaluating the product as it evolves through continuous development.
5. Motivated individuals	An environment needs to be provided that both motivates the team and trusts them to do their job in support of the objectives set out. Thus the introduction of self managed, self governed

	teams into the organization.
6. Face-to-face	Teams needs to be collocated and have easy physical access to the business.
7. Working software	Working software needs to be delivered very early in the life cycle even if that software requires special hardware, data or infrastructure.
8. Sustained development	The project needs to be planned and executed in a way that means effort is constant and that does not rely on any heroic efforts by the team.
9. Technical Excellence	Everyone in the team is responsible for the overall design and architecture and therefore need to have the skills to build elegant architectures.
10. Simplicity	Some of the complexity and effort required for a project is to provide clear communication of the intent and solution of the system being developed. Avoiding doing this work, to keep the project simple will cause problems in the long term.
11. Emergent Design	That all the elements of the project will emerge rather than be designed and reviewed. Thus formal architecture and design reviews will be avoided because of the need for simplicity and because good design will emerge.
12. Reflection	That the team has power to change the things that they perceive to be wrong and thus improve their organization for the future.

If you look at these characteristics you would be forgiven in thinking that agile development means: a collocated, technically excellent team, working in flexible environment, close to their business with no external pressures being applied to them. In an ideal world agility does mean that, but agility happens in the context of a non-agile world.

The Essence of the Principles

The ideas behind the manifesto were born out of an evaluation of the most successful software development projects. These ideas and principles if applied will either make a project very successful or cause so much friction inside the organization that the associated heat will cause the project and often the agile program to be killed. To effectively manage the friction and the adoption of agility it is important to focus on four key ‘essential’ characteristics of agility.

- **Delivery and Change** – The project must continually be in the process of delivery and not worry about getting things wrong and having to make changes to that delivery. Agile projects follow the maxim ‘normally it is better to make a decision and get it wrong rather than not make a decision at all’.
- **Role of the Customer** – There needs to be a special ‘trusting’ relationship between the business and development team. The business must be empowered to make decisions and the team must be able to quickly and easily ask that of the business.
- **Teams** – The team should be self managed and self directing. They must be in charge of their own destiny, deciding what the right work is to do, how they organize to do that

work and buy into the objective of that work. A highly motivated team is by far better than a well organized, well equipped de-motivated team.

- **Technical Approach** – The best architectures, requirements and design are simple. The team must focus on keeping things very simple rather than building perfection.

SCRUM

Scrum is currently one of the most popular agile methods. It is described as a light-weight project management approach. It is named after a particular game segment of Rugby where both teams provide a set of people who have a set of interlocking positions. Their objective is to take possession of the ball. Players on both teams have set jobs, a common objective and are very motivated. The ball is put in to the Scrum and both sets of team try anything to get it by either ‘hooking’ the ball or by pushing-over the opposition. The analogy holds true for development teams – Common objective, focused motivated team, everyone has skills, but rather than everyone having detailed plans a very light set of rules allow Scrums and software development projects to be very effective.

The lifecycle for Scrum is simple. Teams select items from the product backlog and deliver them in a “sprint”. The sprint is short, fixed period of time where all the work happens. The objective of the sprint is defined by the team.

The team meets every day in a short ‘scrum meeting’. The objective is to inform the team as to what work was completed, what was planned to be worked next, and any obstacles in the way.

There are three roles described in Scrum. The *Product Owner* is the empowered business representative who can make decisions and works very closely with the team. The *Scrum Master* protects the team and provides the things the team needs to be successful. The final role is the development team itself.

Scrum is not the only agile method. There are many different methods in circulation today, all with strengths and weaknesses and all would add value to any development team. But, the focus of this paper is on Scrum as an agile method providing a framework that can be applied to other methods and approaches.

Issues with Agility

At face value Agility appears to be very good, but there are 3 main issues which often undermine the ability of a team to apply this agile approach:

- Customer-driven delivery is good, but what about the other risks such as architecture and integration with other projects.
- How does management gain visibility? Project risk needs to be effectively managed to ensure that funding is being apportioned correctly and that the project is moving in the right direction. Having executing code provides some understanding of progress but does not, for non-technical people, provide insights onto the underlying project progress, cost and issues.

- What about other deliverables that are required? Compliance, audit support and other key deliverables are in place to ensure that when the team moves on to other projects there is some history for the project.

The Three Perspective of Process

People talk about software development approaches as though all approaches are comparable. For instance, XP is comparable to RUP, or DSDM is comparable to SCRUM. But most approaches start out in life as a set of practices in response to a set of problems. For instance DSDM was developed around the need to improve the relationship with the business and deliver software faster. It is often the case that each particular approach has some great ideas, but to make it complete a number of other ideas are added to ensure that the reader of the process is not left thinking ‘this doesn’t work because x is missing’.

The three perspectives of process provide one way of looking at processes and evaluating them in the context of the problem they are trying to solve.

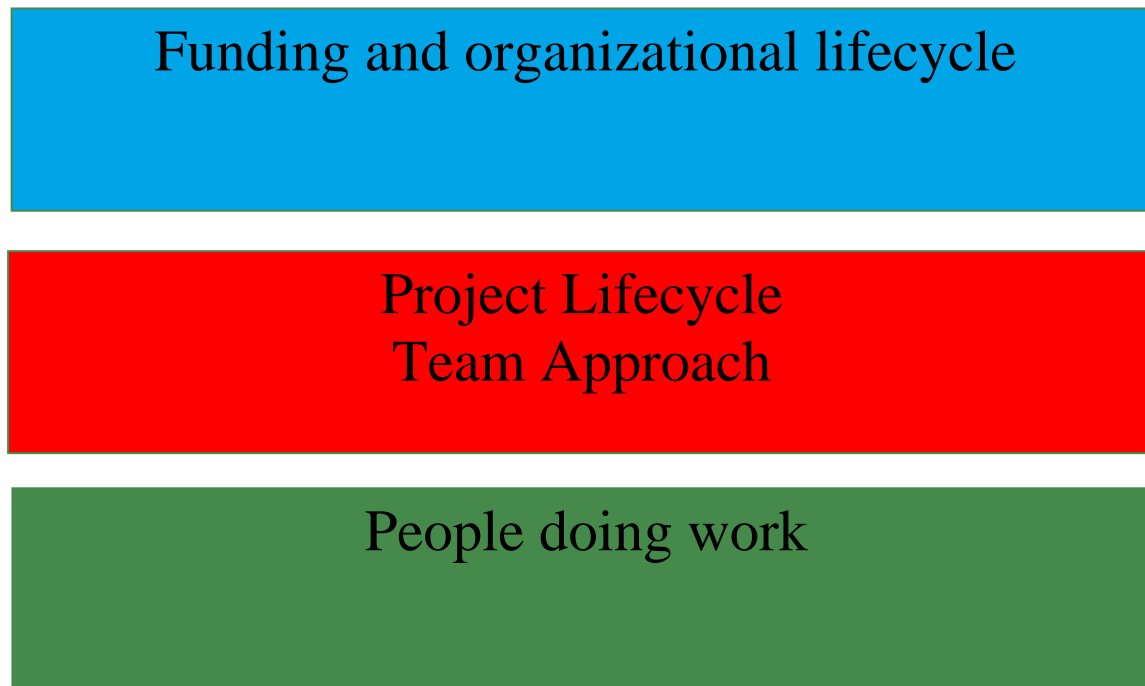


Figure 1 – Three perspectives of process

Figure 1 illustrates three particular views which you can apply to any given software development lifecycle. Each view has a set of requirements and mechanisms for realizing those requirements in a particular instance of the process. The next section describes the requirements, a selection of mechanisms that can implement those requirements and an example of that perspective. The examples draw on an integration between the Unified Process lifecycle, a SCRUM team approach and a Use Case driven technical practice.

Funding and Organizational Lifecycle

This should be the focus of the software development lifecycle of any organization as it provides the ability of that organization to effectively fund, manage a portfolio of projects and report on the progress of those projects. For example looking at the process from a funding, risk and compliance point of view and defining which artifacts, activities and roles are crucial for those requirements to be fulfilled.

Aspects

- Funding – The fundamental driver for this lifecycle is to effectively manage the funding to projects or programs. Thus any project or program must provide evidence that the funds are being spent wisely and in accordance with the policies of the organization.
- Risk Management – To effectively manage a business it is important that surprises are kept to a minimum. This requirement talks to the need for each project or program to effectively manage risk, reporting on and mitigate it.
- Status Reporting – In support of the funding and risk management requirements it is important to provide clear statuses on what is happening in the projects to ensure that ‘collisions’ and other cross-project issues do not occur.
- Portfolio Management – In any large organization projects often fit into much larger more complex initiatives. Each project that is included needs to provide some sort of mechanism to effectively enable integration between those projects.
- Change Management – Documenting what has happened and providing the ability for changes to be undone is crucial for any organization.

Mechanisms

- Milestones – By providing a clear set of milestones each project can report on progress and completion of those milestones. Examples include definition of scope, risk mitigation, architectural POCs, and so forth.
- Artifacts – Most Software Development Lifecycles place too much emphasis on artifacts as a way to manage progress and risk. Artifacts can be a very valuable way of communicating various aspects of a project, but should be accompanied by other proof points as to progress and risk mitigation.
- Meetings – Often combined with the artifacts are set meetings where a pre-selected group of people attend and sign off status. These meetings can be very effective if they are the right people and the artifacts are in the right form that allow those people to quickly and easily participate. .
- Plans – In particular the high level project plan. Having a plan is crucial, but having a detailed plan that is not correct is very misleading. It is important to have a high level plan which is both simple to communicate and change.

Example of a lifecycle

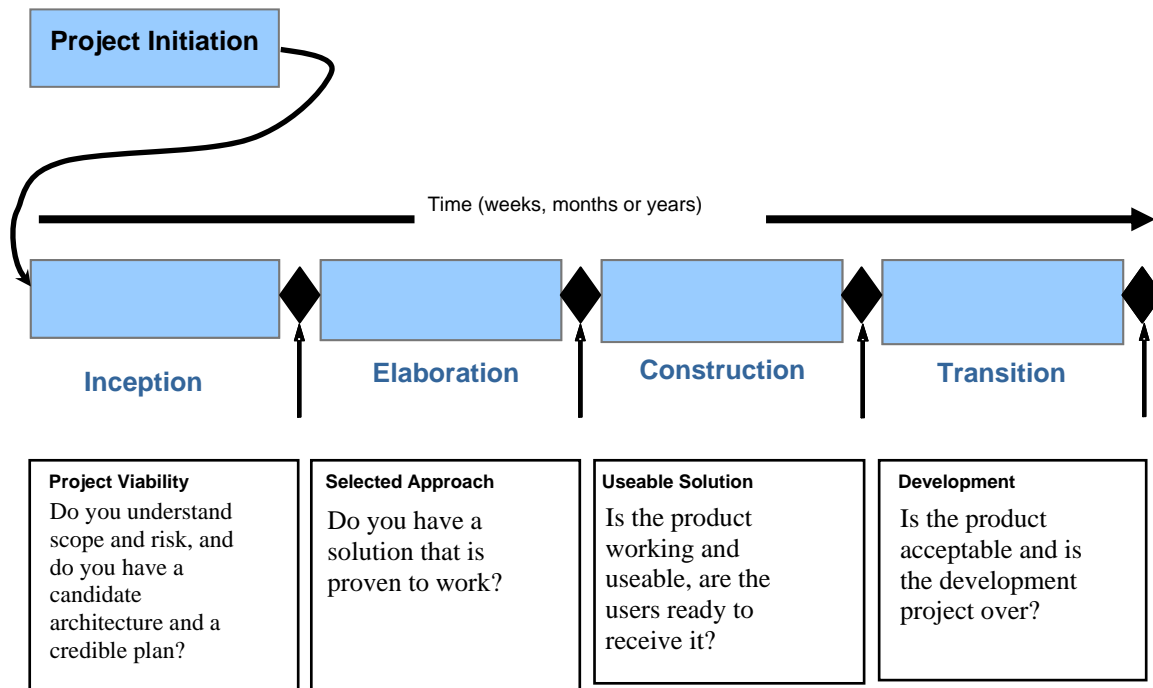


Figure 2 Simplified Unified Process Lifecycle

Figure 2 describes a simplified version of the Unified Process Lifecycle. Each phase has a clear set of milestones, artifacts and can include a set of meetings. The objective of the phase is defined and thus allowing the project to clearly describe progress against that objective.

Project Lifecycle / Team Approach

The team perspective focuses on the necessary things that need to be in place to effectively enable, motivate and manage the team. This perspective focuses on the team level activities, responsibilities and artifacts.

Aspects

- Team Organization – Who does what in terms of responsibilities and work?
- Organizing work – How the work is structured, for example; does each person take one item and build out the requirements, design and implementation, or would it better to break down a requirement into particular disciplines? Supporting this organization is a plan that describes what work is going to be done, by who when.
- Status Reporting – How frequently and what form does the status reporting come in? Typical SCRUM projects report status against tasks in a daily form with overall progress being described in the form of a burn down.
- Skills – Determine what skills the team needs to do its job?
- Efficiency and Effectiveness – How does the team measure how effective it is, and put in place plans that would enable it to improve?

Mechanisms

- **Task** – The allocation of work items, or tasks to the team is one way a team can manage the work and its allocation. A task is owned by a team member, its relationship with other tasks is held in the team plan. It is also possible to associate other actions with the task such as checking out artifacts in the change management system, or entering time in time sheets.
- **Team Meetings** – The primary mechanism to manage progress is often team meetings. These meetings report status and progress, highlighting any issues and problems to the team.
- **Micro Plan** – The team work around a plan. That plan lists tasks, estimates, dependencies and who is responsible for which task. The plan provides the project manager with a mechanism to ensure that the team is on track and making progress. As each task is accomplished the plan is slowly completed.

Example of a team approach

Standard Scrum sits very nicely within this perspective providing a clear set of light weight management techniques that enable the team to effectively work together, deliver work and interact with the business.

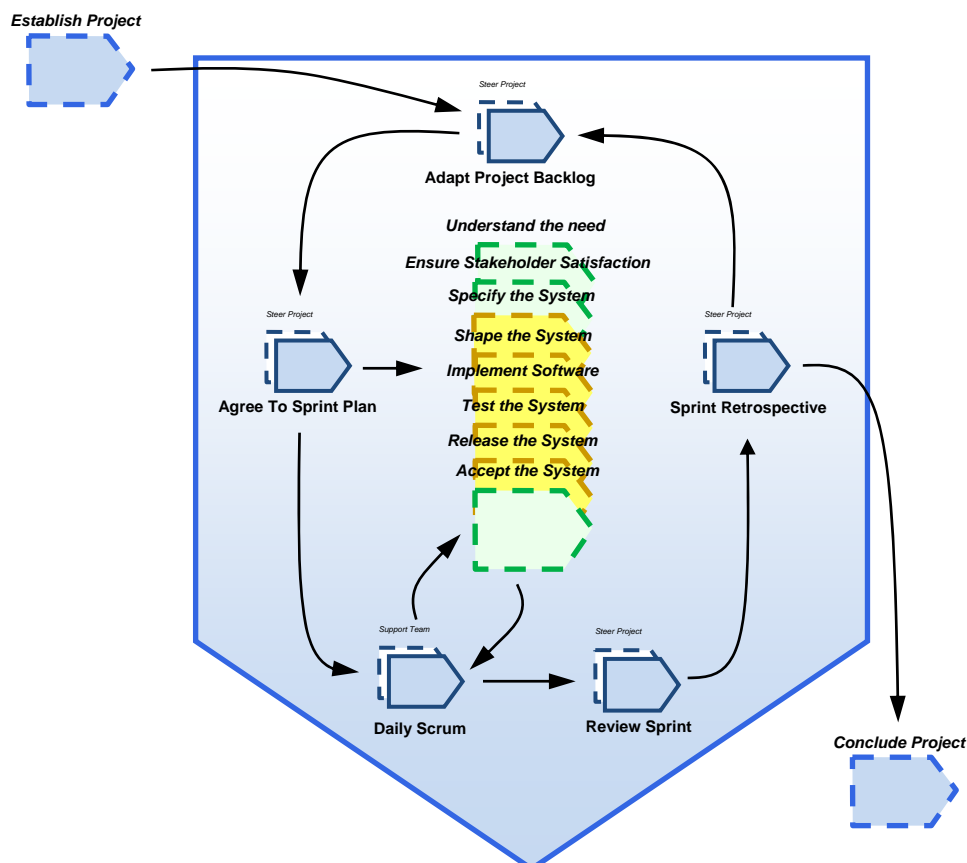


Figure 3 – SCRUM team lifecycle in EssWork notation

The Scrum lifecycle provides a very strong set of mechanism to enable the team. Work captured in the project “backlog” is structured into sprints. A sprint is a time-boxed period of work, with a clear objective. These sprints focus on a subset of the project ‘backlog’ called the sprint backlog. This backlog is committed to by the team for this sprint. The team then plans that sprint out, allocating the work to the team and as they, the team, have decided. The team meets every 24 hours to report status against the tasks allocated describing progress and issues. This information is described on a burn down chart as progress is made within the sprint. At the end of the sprint the team demonstrates the objective of the sprint and reviews how they worked within this time box.

An overall lifecycle (as depicted in Figure 2) provides context to the sprints that the team is working on. The sprints will undertake work that moves the project through the milestones; they will help the team make decisions about which things to do first, ensuring that the overall project is mitigating risk and delivering the work products that are necessary within any organization. The introduction of a lifecycle also helps management manage multiple projects by providing a mechanism by which to compare projects.

People Doing Work – Practice Approach

Another key part of any process is the guidelines it provides supporting people actually doing the work. Processes such as the Unified Process spend lots of time describing techniques such as Use Cases, model driven development and architecture centric design. The majority of agile methods include some details on techniques such as pair programming, user stories and test-driven development.

Aspect

- Techniques - The ability to improve the way the team does the work is key. Most teams have a mix of people from many different backgrounds with different experiences and skill levels. It is important to create an environment that allows team members to use the skills they have, but also encourages conformity and consistency.
- Tools – Tools automate parts of the process providing improvements to consistency and communication. They also may improve the efficiency of performing a given task through the application of automation. It is crucial that the tools are consistent with the techniques being applied and the skills the team have. For example having a modeling tool to build the service interfaces would only be appropriate if the team knows how to model service interfaces!
- Support – Having people that the team can call on when they are stuck or need some advice has a massive positive impact on both productivity and consistency.

Mechanisms

- Engineering Techniques – Training people of engineering techniques associated with software development is often the best way to ensure consistency and control of the technical practices.
- Mentors and Support – This is also known as a community. The establishment of a community provides mechanisms to assign mentors, run training and support sessions,

share best practices and more importantly learn who is doing what within the organization.

- Process / Practice guidance – The traditional focus of an SDLC that provides descriptions of the activities, artifacts created and roles needed.
- Tool Examples and Templates – Providing a clear set of templates within a tool geared towards the standard practices provides clear guidance to users of that tool. Complementary to that are examples. Examples often are the best way for a practitioner to see how to do something. Learning by example is very effective.
- Pre-built frameworks and canned architectures – Reducing the complexity of the problem being solved and the amount of new code being written through the application of pre-built structures improves productivity and quality of the application. But these architectures have to be robust enough to support many application usages and easy to apply.

Example of an Engineering Practice

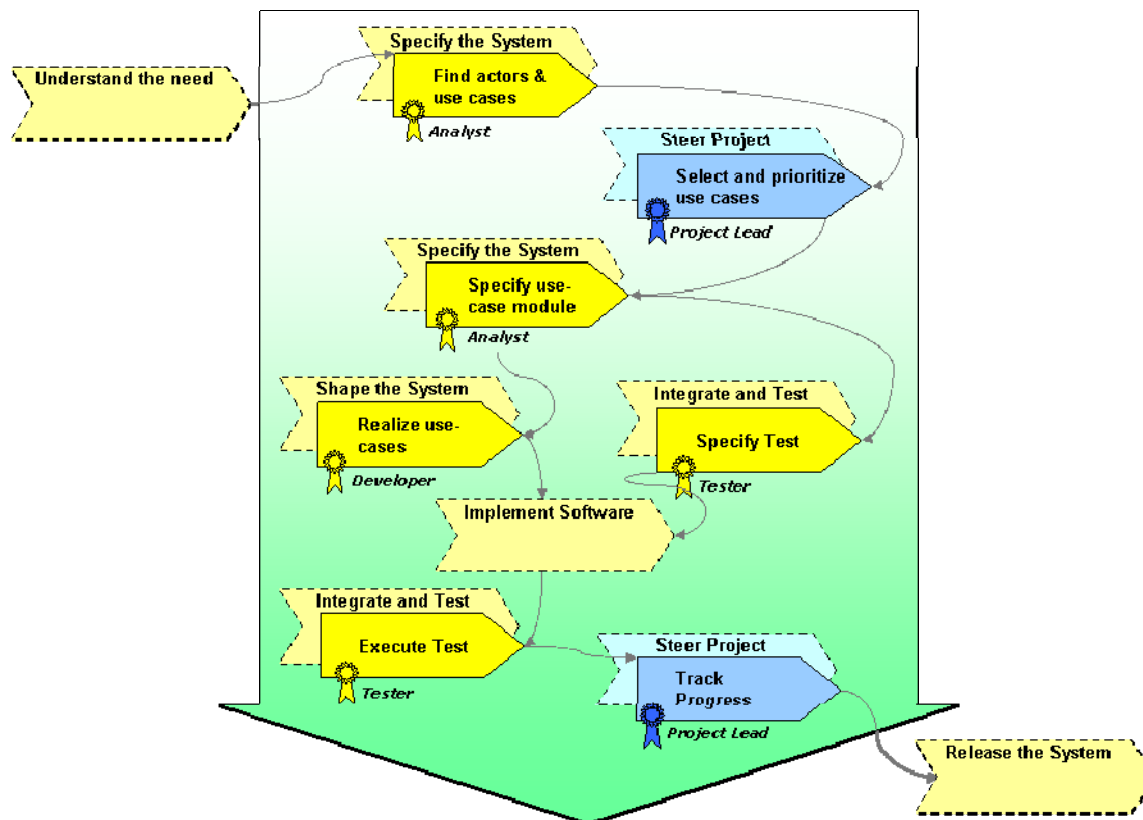


Figure 4 – Technical Practice for Use Case Driven Development

The figure shows our Essential Use Case practice. This practice provides the necessary technical detail to enable teams to develop software using Use Cases as the primary way of describing requirements. Supporting this practice is training and tools to help the practitioners implement this practice in a consistent and repeatable way. The practice would be used to help the team

decide on what tasks are going to be undertaken within a sprint by matching the state the project needs to be in as described in the lifecycle to activities needed to move the project to that state. The resulting list of tasks would then seed the sprint. The application of a series of technical practices, which this is one example, provides the team with some boundaries ensuring that all the teams will develop a similar set of artifacts across many agile projects.

The technical practices do not need to be written down formally and delivered in a structured electronic form. They can be in the heads of the practitioners in the form of tacit knowledge, examples and previous projects. However, the benefit of writing down the engineering practices is that doing so makes the practices explicit so that they can be discussed and compared and improved, providing for better results and consistency.

It is important that you do not force the development team to adopt a set of technical practices, but instead make them appealing either via training, pre-built materials or support. Allowing a team to choose their own practices greatly increases the likelihood of success by creating commitment. Top-down enforcement of practices rarely results in significant commitment to change or improvement, but merely minimal compliance with required processes.

Summary

Agility provides a strong set of social engineering concepts that will, if applied correctly, improve the way in which development teams work. Scrum, for example provides a great set of ideas for both working with the business and organizing the team, but agility in isolation does not work. Organizations must complement agility with both a lifecycle and a set of engineering practices that enable organizations to both better manage the collection of agile projects, but also ensure consistency and repeatability within those projects. Software development lifecycles should avoid being overly prescriptive, focusing instead on the essential states that a project must go through and report progress against. The software engineering techniques must blend into the skills and experience of the teams providing just enough rigor to ensure consistency without providing too much detail which reduces creativity and innovation.

About the Author

David is managing director for North America and has worked in the software development industry for over 15 years holding a variety of posts ranging from developer, team leader and architect. He managed the development of the Rational Unified Process for two years releasing four new versions of the product which including the re-architecting to conform to SPEMM, inclusion of CMMI and Agile content and a revised business modeling discipline.

Prior to that role David worked, in the UK as a consultant in many organizations introducing process and tools. Customers included, Barclays, Merrill Lynch, American Express and the Royal Bank of Scotland. After running the RUP team David took over responsibility of the solutions group at IBM Rational developing content, process and tool solutions in the area of enterprise architecture, CMMI, compliance, and geographically distributed development and outsourcing. This work took David into the area of industry solutions with his final role at IBM Rational as the executive responsible for the Financial Services Sector where he worked with

large Finance customers such as Citicorp, ABN Amro, Fortis, Rabo Bank, LTSB and Credit Suisse.

David is co-author of “Head First Object Oriented Analysis and Design”.

About Ivar Jacobson International

Ivar Jacobson International provides worldwide leadership to help organizations deliver the right solution to your business. Our combination of great people, innovative techniques and technology and customer success make us unique. We combine the right combination of tools, training and mentoring to ensure a consistent, sustainable approach to software development and therefore successful delivery of software based solutions.

For more information, visit www.ivarjacobson.com or contact us directly:

<u>Regional Office</u>	<u>Email</u>	<u>Phone</u>
Americas	us-enquiry@ivarjacobson.com	978-649-2856
Australia	info@ivarjacobson.com.au	+ 61 2 9994 8993
China	info@ivarjacobson.com	+86-10-62091361
Scandinavia	info@ivarjacobson.com	+46 (0)8 501 64 170
Singapore	info@ivarjacobson.com	+65 9772 3538
United Kingdom	info@ivarjacobson.com	+44 (0)20 7025 8070