Position paper: "Are Agile Methodologies Really Different?"

Andy Schneider
as@bjss.co.uk

Despite significant commonalities between agile methods there are also many real differences. Intuitively there have to be significant commonalities between agile methods because, to be considered "agile", the methods satisfy a common set of criteria.

This paper is concerned with differences and not commonalities. There are many significant differences between agile methods and this position paper seeks to enumerate some of the more obvious ones.

Agile methods differ in terms of the requirements they are trying to satisfy, the scope of the solution proposed, the underlying practices described, the degree in which the methods are considered atomic units and the explicit degrees of freedom supported.

Agile methods differ in the requirements they satisfy:

- XP small, co-located teams.
- SCRUM, not limited to small, co-located teams.
- Crystal Clear, amongst other things, clearly targets "Up to 6 developers, typically 3-4"
- Crystal Orange targets projects up to 40 developers.

Agile methods differ in their scope:

- XP prescribes a set of engineering practices within a very lightweight project management framework. It is hard to imagine embedding another method *within* XP.
- SCRUM focuses on providing an agile management framework rather then specifying particular engineering practices. You can embed XP practices within SCRUM (XP@SCRUM) or, for example, use Function Point analysis as the estimation model.

Agile methods differ in their practices:

- XP uses a metaphor to define the architecture, giving the team a common language around which to structure the system. XP does not require a common object model.
- Crystal Clear requires a common object model. This model can be in the form of drawings and/or words. However, the emphasis is on a tangible artefact rather than design stored in the code and oral tradition of the project.

Agile methods differ in the degree in belief that the whole is greater than the sum of its parts:

- Kent Beck advises that newcomers to XP use the whole method, since it is the combination of all the practices that add most value.
- Alistair Cockburn encourages uses of Crystal Clear to select the practices and level of detail according to the requirements of the project. There is less emphasis on the whole creating some synergistic entity that delivers maximum value.

Agile methods also differ in the degree of scaling they explicitly support.

- Most literature on XP describes how XP practices facilitates rapid development of the software and provides advantages to developers within the standard XP model. Very little time is spent discussing how XP projects interface into the rest of the organisation and how formality can be increased as the team scales or organisational requires interfere with "vanilla XP".
- Crystal Clear has a well defined set of work products and techniques and explicit discussions on increasing or decreasing formality depending on project circumstances. It could be argued that whilst this approach offers less promise of a step change in productivity through application of the whole, it provides greater guidance to users of the method than XP when faced with a change in scale or organisation requirements.

The above examples show that the despite the commonalities between methods there are significant differences. These differences express themselves in the different degrees of freedom explicitly supported within the methods. Degrees of freedom help when applying an existing method to a project within an organisation because of the many factors that need to be considered when doing so. Examples of factors to be considered when utilising a development method are:

- Existing programme management, project management, control and governance structures and processes.
- Mandatory requirements placed on the project by the organisation the project is embedded in.
- Size of the project team.
- Skill of the project members.
- Complexity of the project.
- Size (e.g. in function points) of the project.
- Number of organisations the project interfaces to.
- Complexity of deployment environment.
- Non-functional requirements.
- Cost of system failure.
- Politics within the organisation.
- Amount of change the organisation can tolerate.
- History of the project (if already established) or its predecessors.
- Etc.

Given the problem of selecting a method is characterised by so many dimensions it is hard to see how one “theory of development” can satisfy all the constraints. Methods do not have to be unique to have differences. Therefore, whilst “It is easy to argue that the differences between the methods come from different metaphors or backgrounds rather than from inherently unique solutions.”, understanding the many differences between the different agile methods improves the chances that the team may choose an appropriate method for the job in hand and tailor it in an appropriate way.

HOW DO XP, SCRUM AND ASD BUILD THE RIGHT SOFTWARE?

Angela Martin, Robert Biddle, James Noble
Victoria University of Wellington, New Zealand
angela at mcs.vuw.ac.nz

This position paper explores a particular area within software development: requirements analysis and management. We specifically focus on only three agile methods, XP, Scrum and ASD. These methods all highlight, in their respective books [1,2,3], the requirements difficulties often encountered in traditional development. The next sections cover how XP, Scrum and ASD address these difficulties.

EXTREME PROGRAMMING. There are no new practices in XP, it simply takes existing practices and turns them up to an extreme level. The revolution is making the practices reinforce each other. XP faces requirements difficulties head-on by recommending:

(a) The customer is an integral part of the team and should be on-site with the team
(b) The customer writes user stories and then discusses each requirement directly with the programmers
(c) The customer is responsible for *all* business decisions including prioritising user story development
(d) The small 2-3 week iterations allow the user to evolve their requirements based on concrete working software
(e) The customer regularly tests the software to confirm it works as expected

SCRUM. Scrum is a project management method; it aims to provide a mechanism to control the chaos surrounding the communications between the business and the project team. Scrum streamlines the requirements communication:

(a) There is one central prioritised list of all requirements, the *product backlog*
(b) The product backlog is controlled by one person, the *product owner* , however many people contribute to the list
(c) The 30 day sprint allows the project team to focus on a frozen set of requirements, while allowing the business to change their requirements after a tolerable delay
(d) The software is reviewed at the end of each sprint to ensure ongoing feedback to the project team and the business

ADAPTIVE SOFTWARE DEVELOPMENT. ASD is based on complex adaptive theory and describes the mind set and principles required to succeed in developing software iteratively and incrementally. ASD recommends the following for requirements:

(a) There are a set of artefacts (project vision, data sheet and specification) that ensure a shared project vision exists.
(b) Collaboration techniques are used to evolve requirements: JAD sessions, Customer Focus Groups and finally post-mortems or process improvement reviews.
(c) A *collaboration facilitator* role is introduced to focus on "thinking" about and planning collaboration rather than simply "letting it happen".

SO, HOW DIFFERENT ARE THEY? There is significant cross-over between the methods, including:

- The product backlog, user stories and product specifications all ensure there is a centralised list of requirements
- The on-site customer role, the product owner role and the JAD/CFG sessions all aim to ensure the developers do not need to deal with conflicting requirements, including the prioritisation of requirements
- The small and regular iterations ensure there is opportunity to learn and to evolve requirements.

So, while there is some difference in the details of the methods, many of the practices could potentially be interchanged. One of the key overall differences between XP and Scrum is their emphasis on different aspects of software development; XP emphasises programming, while Scrum emphasises project management. XP provides reinforcing practices for developers and Scrum provides comprehensive management practices for project managers. ASD concentrates on outlining learning and collaborative techniques and theories for all project members, and introduces a collaborative facilitator role that specialises in collaborative and learning based communication, including the communication between customers and developers. Both XP and Scrum appear to under-emphasise the challenging job [4] facing the *on-site customer and product owner* and in gathering and prioritising the requirements for the project. ASD begins to address this issue by recognising the need for structured collaborative techniques and roles in this area.

REFERENCES

1. Beck, K. *eXtreme Programming Explained: Embrace Change*, Addison Wesley, 2000.
2. Schwaber, K., Beedle, M. *Agile Software Development with Scrum*, Prentice Hall, 2001.
3. Highsmith, J. Adaptive Software Development: A Collaborative Approach to Managing Complex Systems, Dorset House Publishing, 2000.
4. Martin, A., Noble, J., and Biddle, R. *Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Giancarlo Succi (Ed.), "Being Jane Malkovich: a Look into the World of an XP Customer". Lecture Notes in Computer Science 2675, Springer-Verlag. 2003.

Christian Sepulveda
cs@atdesigntime.com
OOPSLA 2003 Workshop
"Are Agile Methodologies Really Different?"
Position Paper

Each agile process can be characterized by one or more activity biases. For example, Extreme Programming is a coding centric process. Consider the 12 practices: Metaphor, Simple Design, Automated Testing, Refactoring, Pair Programming, Collective Ownership, Continuous Integration and Coding Standards are directly about coding. The remaining practices (Planning Game, Small Releases, Onsite Customer and 40-hour Week) are about either staying out of the programmers' way so they can code or improving communication about what they are going to code.

Scrum is project management centric. Scrum specifies the length of the iteration and the product backlog; both these elements guide the management of the project.

Comparing the XP and Scrum, note that XP doesn't specify the length of an iteration and provides loose guidance for story management. Scrum doesn't offer much guidance for coding or testing. I think the two different core perspectives mark the differences of the two methodologies, but also make the two quite compatible. (In my experience, XP developers welcome scrum as it doesn't outline many activities for them and provides protection so they can code; for a developer, the use or omission of scrum has little impact.)

Armed with a catalog of agile processes, categorized by the activities they are based on, a project team can elect the process or amalgamation of processes that are appropriate for its context and needs. For example, as the activity biases of XP and Scrum are different, they do not interfere with each other and therefore are compatible.

Certain agile processes are incompatible, as they share an overlapping set of activity biases. Feature Driven Development and XP would probably be in conflict as they both have programming biases. However, Feature Driven Development offers guidance for programming, project management and requirement identification. For teams who require structured support for all these activities, FDD may be a good choice.

Processes that have more than one activity bias are higher in ceremony than the processes that have one activity bias. It is important to identify these distinctions; otherwise a team could choose more or less process support than it needs.

I think process refactoring is a necessary and beneficial activity. A project team should consistently evaluate its progress and process needs. Based on this evaluation, the team should augment and modify its process to accommodate its current requirements. This approach encourages a "just in time" or "on demand" approach to process selection and tailoring.

Knowing the activity bias of each agile process supports process refactoring. Process refactoring requires the identification of the team's shortcomings or process "smells". It has been my experience that these "smells" are frequently complications of some activity in the development lifecycle. For example, XP doesn't provide much support for identifying a release date or the termination of project. For some teams, this a project management "smell". A scrum Burndown Chart could provide the project management that alleviates this need.

Position Paper for the OOPSLA 2003 Workshop "Are Agile Methodologies Really Different?"

Jens Coldewey
Coldewey Consulting
Curd-Jürgens-Str. 4
D-81739 München
Germany
Tel: +49-700-26533939
Fax: +49-89-74995703
email: jens_coldewey@acm.org
http://www.coldewey.com

Approaches of a Methodology

Besides considering the processes a methodology supposes to use, another interesting topic for studies is the model that underleis the methodology. Though this is closely connected to the value system, the model bears no ethical judgement, as a value system does.

There are two popular models for projects: The control model and the systemic model. The control model assumes that there are centralized steering roles in a project, skilled individuals who plan and manage the project. These roles include the project manager, the architect, chief programmer, and so on. With this approach success and failure of the project mainly depends on the performance of these controlling roles. I will refer to these roles as the „managers", though this includes all kind of management and controlling. Methodologies using this approach usually concentrate on elements that support this structure:

- Enhancing communication between the managers and the operational team members, such as programmers.
- Providing reporting and controlling structures to enable the managers to get a good picture of the project status.
- Ensuring that the operational work is done as uniform as possible to ease planning, prediction, and controlling. Team members are assumed to play given roles and to be exchangeable „resources".
- Clear separation of responsiblities.

The ultimate extreme of this appraoch is Frederick Taylors „Scientific Management" that set the ground for the economic boost of the last century. Most traditional processes follow this model.

An alternative approach is to view a project as complex system. Complex systems have some characteristics, command-control structures don't have: Their behaviour is hard to predict and they are very flexible. To contol a complex system doesn't mean to predict and control every single action in it. Rather it means to understand the structure and interrelations of the system and set them up so the the system is able to adapt itself to the needs. Systems Thinking provides the appropriate toolset for this kind of approach.

Methodologies based on systems thinking — no matter whether consciously or not — show some typical properties:

- Few rules. Most of the rules establish feedback loops and communication to ensure all parts of the project are still heading towards the project goal
- Feedback and communication are designed to be as efficient as possible, rather than being trackable or controllable.
- Few if any centralized or hierarchical roles. Team members are assumed to be well-trained, responsible individuals who are able to organize themselves

Comparing the Approaches of Agile Methodologies

The following list checks to which approach the different agile methodologies belong:

- *Adaptive Software Development:* ASD is the only methodology that is explicitly based on the theory of complex adaptive systems. It restricts itself to establishing a feedback loop („Speculate, Collaborate, Learn“) and setting up the environment for efficient project work. A classical representative of the systems approach.
- *Chrystal Methodologies:* The major commonality of all Chrystal Methodologies is a process check workshop at least twice per increment. In addition Cockburn delivers seven principles to be observed in a project. Roles and organizations are only provided as suggestions. Hence we have another example of a systems approach.
- *Dynamic Software Development Method:* DSDM defines explicit roles and responsibilities as core of the methodology. Feedback loops are provoded for the delivery (Frequent Delivery, All changes are reversible) but not on process level. Hence DSMD uses a systems approach to let the software grow but a control model to manage the project.
- *Extreme Programming:* XP uses only a few rules to set up an environment in which software can grow. So the technical part clearly uses a systems approach. Concerning the methodology XP showed an interesting development in the last years. Though Kent Beck's metaphor of „Learning to Drive“ pointed towards a systems model, initial statements like „Either you do everything as written or you don't do XP“ were often interpreted as a sign for a control model. Figuring out that these statements drove XP into an unwanted direction, the community now agrees upon regular adaptation of the process. Therefore, XP may be considered a systems methodology today.
- *Feature Driven Development:* FDD installs the role of a Chief Programmer and uses upfront designs. Hence, FDD in my perception clearly uses a control approach rather than using a systemic view, even though it definitly is a light weight process.
- Lean Development: Most of the technical LD practices are similar to what XP found useful, so technically LD is also based on a systems model. The management is highly focussed on self-organization and feeback — indicators for a system model too.
- *Scrum:* Defining a central feedback loop as major control instrument identifies Scrum as systemic methodology. This adds to the fact that scrum defines few roles with the central role, the „Scrum Master“ having responsibility mainly for facilitating the process instead of the deliverable.

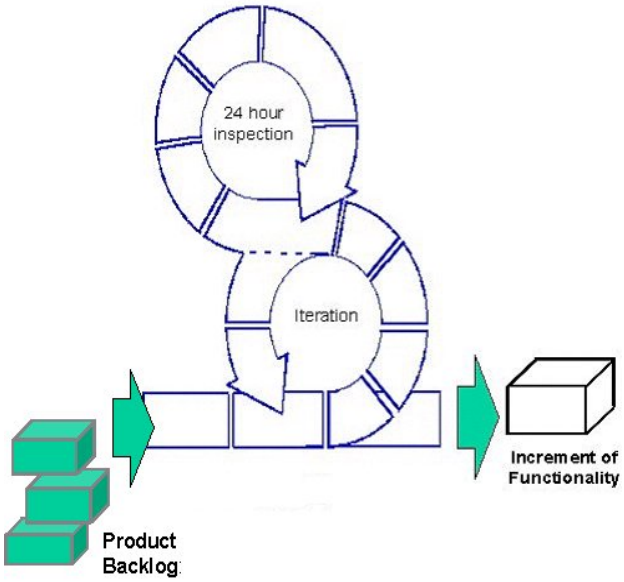
A Provoking Conclusion

I suggest to define agile development it as a methodology based on a systemic view on software development. This view leads to the agile principles as well as to many agile methodologies.

Using this definition would mean to re-evaluate the current understanding of which methodologies are agile and which aren't. According the above list this would narrow the „fully agile“ methodologies to ASD, Crystal, XP in its current form, Lean Development, and Scrum. DSDM would be considered as „technically agile“ methodology and FDD would be seen as a lightweight traditional process.

Are Agile Methodologies Really Different?
Ken Schwaber, ADM, for OOPSLA 2003

Agile processes have a common basis in that all of them accept the premise that software development is complex and that mechanisms other than predict and hope are required to manage the complexity. A common mechanism for addressing the complexity is iterative, incremental development, which I refer to as the agile skeleton. All agile processes share this skeleton, fleshing it out with different practices based on the orientation of the originators of the process and the domain in which it is mostly applied.



Scrum is based in empirical process control. The three mechanisms applied within Scrum are visibility, inspection, and adaptation. The skeleton supports these. A prioritized, emerging list of requirements is kept visible so that everyone understands what will be done next. Every twenty-four hours the work of the team is made visible and inspected by the entire team at a short meeting called the Daily Scrum. At the end of each iteration, the product, or increment, of the iteration is made visible and inspected by all stakeholders at a Sprint Review meeting. Every time an inspection is made, adaptation is called for. At the Daily Scrum, the team members adapt to each other's progress. At the Sprint Review, the stakeholders adapt to the progress represented in the increment by selecting what they want the team to do next (stakeholders indicate what to do next, the team chooses how much of the what it believes it can turn into an increment).

Visibility implies that everyone knows and agrees upon what they are looking at. This introduces the concept of sashimi, a slice of the whole equivalent in content to all other slices of the whole. For the Daily Scrum, the slice of sashimi is a report that something is done. "Done" implies accepted engineering practices that indicate that done means coded. Or maybe it means coded, unit tested, checked in, built, and acceptance tested. Either way, there must be a common understanding; otherwise mistakes may be made as various team members inappropriately adapt based on incorrect assumptions of what they inspected. Similarly, the increment created every iteration must be well-defined and similar every time. In Scrum, each increment is an increment of potentially shippable product functionality. If a stakeholder inspects demonstrated functionality, they can presume that it is fully developed, clean, refactored, fully tested, and with only a small amount of packaging can be implemented.

The last aspect of agile processes is the heart. This occurs within the skeleton's iteration. At the OOPSLA'02 workshop on commonality of agile processes, we decided that creativity, the ability to figure out the right thing to do and then do it, is common and the heart of all agile processes. Rather than someone else telling the team what to do and how to do it, the team is free to devise the most appropriate engineering practices and approaches to turning requirements into functionality.

The skeleton queues up the most important work for the business every iteration, maximizing the return on investment of the project. Sashimi ensures that progress is inspectable. The heart maximizes the productivity of turning the requirements into functionality.

I've been conducting training sessions over the last six months during which I help people who understand Scrum, Extreme Programming and other agile processes to understand the nuances and ins-and-outs of Scrum. I have been struck that the people who know XP through years of practice see how complimentary Scrum and Extreme Programming are. Both are iterative and incremental, both aim at improving the lives of the team members, and both focus on improving the quality of the product. Extreme Programming has even adopted the daily standup from Scrum's Daily Scrum meeting. People have told me that they see Scrum as the management approach to agile development and XP as the engineering practices that make it effective, both bonded together by complimentary practices and goals.

The differences that I've noted between the two as I've explained Scrum to XP users are:

1. Scrum is more coarse grained. The Product Backlog items are coarser than stories, and the iterations are coarser at thirty days than more XP iterations. The reason for Scrum's coarse grained practices is to keep everything in easy terms for the user, stakeholders, and customers to understand. XP stories are more specification and feature oriented, Scrum Product Backlog is more user functionality oriented.

2. Scrum requires the team members to learn how to talk to the customer without involving technology. XP seems to require the customer to learn how to talk to the developers in terms of specifications.
3. Scrum's teams are more self-organizing and self-managing. The ScrumMaster is not allowed to tell the team how to do its work, whereas the Coach is much more active in directing the team's work.
4. Scrum is much more the art of the possible, where customers, stakeholders and users are expected to inspect what a team has been able to do and make the appropriate adaptations. XP focuses more on the precision of estimating with "yesterday's weather" and velocity, looking to show the customer that they can be trusted to know what can be delivered. Scrum delivers what it cans and places the responsibility on management to then figure out what to do.
5. Scrum can be implemented in days, and XP takes longer.
6. The name Scrum is just kind of gross, but the name "Extreme Programming" is frightful.

I'm sure that the workshop will uncover thousands of other variations and similarities, but overall I find their consonance to be greater than the discord. Accordingly, whenever I have a situation in which software is being developed, Scrum is being used, and the engineering practices are substandard, I also call for XP to be implemented.

Agile Ecosystems

A Position Paper for the Workshop: Are Agile Methodologies Really Different?

by Mary Poppendieck

Jim Highsmith identifies seven agile software development ecosystems[1]:

1. Scrum
2. DSDM
3. Crystal Methods
4. FDD
5. Lean Development
6. XP
7. Adaptive Software Development

One of the big differences between these approaches is the amount of specific guidance vs general guidance embodied in each. Scrum, DSDM, FDD, and XP give specific guidance in the areas they cover, while Lean Development, Adaptive Software Development, and Crystal Methods present a theoretical basis for agile practices.

Taking on the theoretical methods first, each has a somewhat different focus: Crystal concentrates on communication and varying practices based on project size and risk. Agile Software Development focuses on emergence, while Lean Development emphasizes the traditional lean concepts of value and flow. All three emphasize the primary importance of the development team. None of these three methods focus on specific practices, so they do not find themselves in conflict with the four agile methods that offer more specific practices, or with each other, for that matter.

The four other agile methods: Scrum, DSDM, FDD, and XP offer specific practices which are generally expected to be followed as a package until one is expert enough in the method to modify the practices. DSDM and Scrum focus on project management and are agnostic about the underlying technical approach. Both emphasize business value, fixed timeboxes, significant customer involvement, developing high priority items first, and stopping when you run out of time. DSDM is the higher ceremony approach, while Scrum spends less time on project initiation activities.

FDD is a complete approach covering both technical issues and project management. It is different from the other agile practices because it does not advocate common code ownership, but rather assigns class ownership. This rather large technical difference underlies many of its other differences from other agile practices. Because of its comprehensiveness and different technical approach, FDD would not co-exist well with other agile approaches, but would be used separately.

---

[1] *Agile Software Development Ecosystems,* Addison-Wesley, 2002.

Scrum and XP are often used together because their practices are more or less disjoint: Scrum focuses on project management while XP focuses on developer practices. Both of these approaches are often billed as a complete set of practices that are supposed to be used without modification, at least until the users become skilled enough to make appropriate changes. However, when used together, a few differences have to be resolved, including recommended iteration length, specific planning details including release planning and stories or backlog items.

However, for all of their differences, I find the most interesting difference to be the way leadership is viewed in the various practices. Here is how I perceive the way each practice deals with leadership.

1. XP

XP has the role of a coach, but this role is not necessarily considered a leadership role. I perceive that XP is in part a reaction against bad management, and thus there has been a reluctance to create a role that can degenerate into a bad manager. In addition, XP makes little distinction between an XP coach who is a consultant that comes from time-to-time to give advice, and a coach who is involved in the day-to-day activities of the team. XP expects a 'customer on site', and expects this customer role to provide all leadership in terms of definition of business needs and guidance of team priorities.

2. Scrum

Scrum has the role of a Scrum Master, a person who organizes the daily Scrum meeting, runs interference with management, and helps break down barriers. Outside consultants are not generally considered Scrum Masters, they assist Scrum Masters. A Scrum Master is not allowed to help the team figure out what should be done, and does not serve as a technical lead. Scrum also has a role called a product owner, generally outside the team. This person (and the product owner is expected to be one person) defines backlog priorities.

3. DSDM

DSDM identifies specific roles, a key one being the technical coordinator. The technical coordinator is responsible for system architecture and technical quality, as well as basic practices such as configuration management. This role is quite different from the coach of XP or the Scrum Master of Scrum, where technical leadership is left to the team as a whole. DSDM also has a role called 'Visionary', someone similar to Scrum's Product Manager, but possibly less involved. DSDM also has an Ambassador User role, a role similar to the 'customer on site' role in XP.

4. Crystal Methods

I can't say much about Crystal Methods as I am not very familiar with them.

5. FDD

FDD has the following roles:

Project Manager – Administrative lead, responsible for securing funding and other resources and for reporting progress. This role is not like a Scrum Master, but has some similarities.

Chief Architect – Responsible for system design, but more by providing guidance than direction. Similar to the technical coordinator of DSDM.

Development Manager – Responsible for day-to-day development activities. I'm not really sure what this person does, and do not see an equivalent in other agile ecosystems.

Chief Programmers – Provide technical leadership to small teams. Sort of like mini-chief architects for sub-portions of a large project.

Class Owners – Each class is owned by a single developer, who makes all changes to it.

Domain Experts – Well, every ecosystem has to have users.

6. Lean Development

Lean Development looks for several kinds of leadership – Master Developer (similar to the Chief Architect of FDD or the technical coordinator of DSDM), expertise leadership (leadership in specific technical areas such as GUI design, database development, security, etc.) and project leadership (similar to the role of a Scrum Master).

7. Adaptive Software Development

Adaptive software development promotes the leadership-collaboration model as opposed to the command-control mode. Leadership-collaboration focus is on work states rather than process, on creating a collaborative environment, and on creating accountability for results. I'm not sure how to compare this to other roles in other ecosystems.

Conclusion:

True to form, the higher ceremony methods are also more formal on defining leadership roles. The methods which focus more on self-directing teams seem to be more adverse to defining leadership roles, probably worrying that a leader will interfere with team decision-making. I think that there is a lot to be learned from operations management, which has developed good theories on how to balance team independence with good leadership. A few lessons on how to effectively manage a high volume call center or a large retail store might do us all some good.

Mary Poppendieck

What's different about Agile Methodologies?

I believe agile methodologies represent a return to the idealistic notions of people like J.C.R. Licklider, Norbert Wiener, and Douglas Engelbart, who valued computers more for their human capability enhancement than for their economic or military benefits. While early researchers dedicated themselves to removing the time barriers posed by batch interactions with the computer, agile methodologists seem to be working towards removing the process barriers to true interactivity between the problem solver and the computer. This is accomplished by involving customers as full and valued partners on the project team and acknowledging the support of their work as the primary goal of the team's effort.

According to R. W. Terry (2001), our history contains both the seeds of our future possibilities and the limitations of our ultimate potential. Historical context provides the foundation for our action and is one of human action's seven dimensions. Software development practitioners could learn much from Terry's leadership model, which he built on these seven concerns by synthesizing decades of leadership research. Unlike many development models, his does not suggest that progress consists of advancing from one leadership zone into another, but that progress is achieved only when each concern is given its due. Every feature must be addressed in every action to the extent that it is relevant.

As shown in figure 1, relevance is determined by balancing two factors – certainty of outcome and agreement on direction. Agile methodologies are poised at the boundary between chaos and certainty, where meaning must be created – the point at which Terry calls for leadership rather than management. As the certainty of outcome becomes lower and agreement on direction becomes vaguer, agile methodologies become crucial. Their acceptance of the idea that the proper solution will emerge from dealing with concerns as they express themselves builds confidence in the team's ability to deal with uncertainty.

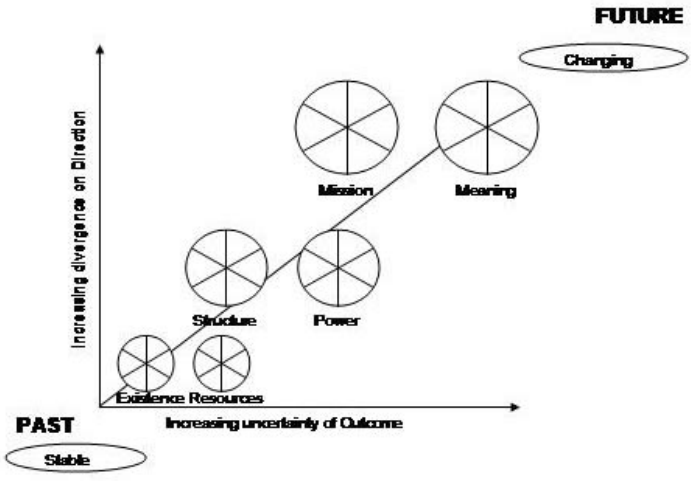


Figure 1 - 7 Zones of leadership (Fulfillment comprises and extends the others)

Each zone comprises aspects of all the other zones, but exists independently as well. Fulfillment represents the ultimate expression that results from integrating the other

six zones, reflecting on the wisdom gained there, and drawing inspiration for further action. Figure 2 shows how Terry illustrates their relationship.

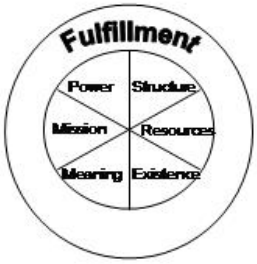


Figure 2 – Action Wheel developed by Robert W. Terry

Terry's seven dimensions, which resolve to 11 zones, are:

Zone	Concern	Description	Software Development Implications
1	Existence	Emphasis on probing the historical background of the present situation	Traditional methods emphasis on repeatability vs. agile emphasis on applicability
2	Resources	Emphasis on building core competencies, tools, scientific method, research, practical experience	Traditional emphasis on tools vs. agile emphasis on people
3a	Structure biased towards form	Emphasis on interconnected sustainable systems, defining roles and responsibilities, building infrastructure	Traditional segmentation of roles and responsibilities vs. agile generalists
3b	Structure biased toward process	Emphasis on affirmation of core values, strategic planning, coherence between what is said and done, team-building	Traditional emphasis on step-wise definition of process vs. agile emphasis on team to figure it out.
4	Power	Emphasis on creating ownership, participation, negotiation, empowerment	Traditional development team vs. agile cross-functional teams
5a	Mission biased towards ends	Emphasis on participatory direction-setting, framing the future, convergence	Traditional emphasis on meeting the spec vs. agile emphasis on meeting the need
5b	Mission biased towards means	Emphasis on developing anticipatory wisdom to deal with change, pattern recognition, metaphorical thinking	Traditional emphasis on pre-determined milestones vs. agile emphasis on continual progress toward customer satisfaction
6	Meaning	Emphasis on creating meaning in chaos through co-creation and centeredness	Traditional methods don't go here; agile methods have made inroads.

7a	Fulfillment focused on wise choices	Emphasis on authentic discernment of problems and application of appropriate solutions	This remains a goal for all development methodologies.
7b	Fulfillment focused on puzzle-probing	Emphasis on tensions among the six previous dimensions, their dependencies, and polarities	This remains a goal for all development methodologies.
7c	Fulfillment focused on living the promise	Emphasis on servant leadership, spirituality, ethics, values	This remains a goal for all development methodologies.

Agile methods represent a foray out of the systems world into the world of complex adaptive systems where accepting the destination motivates the journey, but the journey itself may demand a renegotiation of the final objective. Agile methods look both inward and outward for inspiration. They resonate with Terry's concept of authenticity - "action that is both true and real in ourselves and in the world." Authentic action fosters a resonance between a person or community's values and its surroundings while at the same time ennobling its spirit and engendering authentic action by others "in the commons" (p.112). Actions are authentic to the extent that they both express intent and enable positive growth in the world. The agile emphasis on constant feedback guarantees consistent attention to congruence between the project team's intent and its results. Thus, agile methods, while each addressing slightly different problems, enrich the software development community through their emphasis on underlying values rather than specific tools and through their reliance on people to provide whatever is needed to deliver value to the user.

References:

Terry, Robert W. (2001). Seven Zones for Leadership: Acting Authentically in Stability and Chaos. Davies Black Publishing, Palo Alto, CA.

Terry, Robert W. (1993). Authentic Leadership: Courage in Action. Jossey-Bass Publishing, San Francisco, CA.

Submitted by Pam Rostal as a position paper to the 2003 Workshop on Commonalities among Agile Methodologies

Differences in Agile Methods: A Matter of Conscious Practices or Not?

Rebecca Wirfs-Brock
rebecca@wirfs-brock.com
www.wirfs-brock.com

Agile methods are infused with practices geared towards producing value while eliminating wasteful acts and artifacts. Key to most agile practices are customer involvement, short development iterations, and delivery of working code. Yet there is a profound difference among agile methods (and agile thought leaders) in how much thinking and reflection they encourage.

Christopher Alexander in Notes on the Synthesis of Form contrasts unselfconscious and conscious design processes. Alexander characterizes unselfconscious design cultures as places where:

- Building skills are learned informally
- Builders live in the systems they create
- The unspoken rules are of great complexity and are rigidly maintained
- There is a way to do things and a way not to do things
- There is firmly set tradition which is accepted without question
- Changes are made locally whenever something needs improvement

Unselfconscious building processes result in well-fitting forms that persist in equilibrium with the system they exist within. Structures produced in such a system are, according to Alexander, "not the work of individuals, and their success does not depend on any one man's artistry, but only on the artist's place within the process" The major downside is that when presented with complicated (new choices) unselfconscious processes aren't readily adaptable. Unselfconscious builders solve problems in a narrow, well-known context.

In contrast, in a selfconscious design culture things are done very differently:

- Principles and practices are distilled into systems of concepts. There are schools of thought, teachers who teach them, and students who debate them
- Builders think in terms of abstractions, categorization, and explicit principles
- Buildings are more permanent and frequent repair and readjustment is less common
- Construction isn't in the hands of the inhabitants

As a consequence, Alexander emphasizes that reaction to failure is less direct. Failures are reported and described before they are fixed—but only by specialists. With a selfconscious culture, concepts and principles and practices create blinders that can lead to failure: "concepts control…perception of fit and misfit—until..he sees nothing but deviations from his conceptual dogma, and loses.. the mental opportunity to frame his problems more appropriately."

Of course, Alexander is referring to builders of buildings while we build complex software. There are parallels, but we shouldn't blindly look for them. As software developers we don't have a long culture or tradition of building things repeatedly to draw upon. Our systems are not rebuilt again and again and again. Our materials and techniques are rapidly changing. And so are our practices.

So how do different agile methods deal with selfconsciousness? No agile method is purely unselfconscious. All agile methods have written practices. Fortunately most agile authors write slim, approachable books. But some agile methods clearly provide frameworks for thinking while others offer an integrated set of techniques that are learned largely practiced unselfconsciously. Agile methods can be characterized along several dimensions:

- Being more or less open—where change and adaptation by practicing builders is actively encouraged
- Consisting of practices accompanied by explicit principles and values or not
- Being integrated systems adopted "whole cloth" or collections of practices

- Providing conceptual frameworks for decision-making and thinking or explicit practices and techniques

Some agilists would like practitioners to be mostly selfconscious. Yet many builders of software will never be happy with closed world views:

> One of my issues with the leaders of movements like SCRUM and XP is that they believe that you should be using their techniques verbatim— I once heard one of them say "You're either doing XP exactly, or you're NOT agile!"—a reviewer of a book on Amazon.com

Others become upset that when authors wax philosophical:

> …[title purposefully deleted] is one of the few software books that I would return. It is full of abstract fluff (which is repeated in several different ways). This reminds me of an academic textbook. Just take a simple concept and invent some terminology to wrap around it in several different ways. —another reviewer on Amazon.com

When I first read Alexander's words, I got angry. He was clearly biased towards unselfconscious practices and against forming conscious practices. I tend favor agile methods that encourage critical thinking and exercising judgment within a value framework. But I'm too much of a thinker to blindly do anything without knowing the "why behind it" so I can adapt when necessary! I'm the proverbial two-year old who questions "why, why, why?" I resonate with authors that encourage critical thinking and exercising judgment as well as distillation of their principles and practices. I want to have the best of both worlds.

I caution you (and myself) against placing black and white value judgments on selfconscious or unselfconscious building processes. There is a time and place for both. I don't want to think when I slam on the brakes to avoid a collision. Sometimes I want to think about what I have for dinner and how tastes and textures of food complement each other. Sometimes I don't. Not every act or practice needs to be questioned. Software isn't always like driving a car in rush hour traffic or preparing a gourmet meal. In the software cultures I have been part of, developers freely quibble with various methods and practices. Most always take every practice and fit it into their own context.

I encourage agile thought leaders to be explicit about the contexts and cultures they are intimate with. Say what your method doesn't address as well as what practices are important and vital. In our hyper-connected age, it is hard for someone to follow ideas "correctly" after skimming a book, reading an article, or listening to an entertaining speech. You won't be there to guide practitioners. Yet your ideas are out there making an impact. Think long and hard about whether you want to encourage unselfconsciousness (and what you want to be codified into a self-conscious practice). Think about whether you want to be open and adaptable and explain why. Work hard at explaining both your core values and core practices. When you learn more, share these learnings with others.

To practitioners of agile methods, I offer some questions for sorting out various methods: Do you want a system of practices or a framework for making decisions? How much do you want to add to or adopt a set of practices? How much reflection on your practices can or will you tolerate? Are you looking for answers or approaches?

References

Christopher Alexander, Notes on the Synthesis of Form, Harvard University Press, 1964

Position Paper for OOPSLA 2003 Workshop #2: Are Agile Methodologies Really Different?

Steve Berczuk
Aliaswire, Inc – www.aliaswire.com
steve(at)berczuk.com/www.berczuk.com

My primary interest in this workshop is that I am moderating a panel *What's so eXtreme About Doing Things Right?* (http://oopsla.acm.org/oopsla2003/files/pan-9.html) on Thursday Oct 30 with a similar theme. My tendency is to wonder not what is different between the agile methods, but rather, what the difference is between using an agile method & just following "good" practice. Rather than asking "are agile methodologies really different (from each other)?" I wonder "are agile methodologies really different (from good practice)?"

On any project you are faced with the problem of how to deliver a certain amount of functionality to a customer with a certain level of quality given the constraints of a project (time, money, resources, skill levels of the implementation team, etc). Defining *quality* as "value to some person" (Gerry Weinberg's QSM Vol-1, p 7), your process should also help you and your customer to discover what the software needs to do to create value. To do this, the process should foster communication, and manage the customer's expectations. The best way to foster communication and manage expectations is refine the project based on frequent feedback. Using an agile process just makes explicit a management model that relies on these things. I suspect that most management models rely on these things as well; they are just not explicitly enforced.

All agile methods have techniques to assist with all of these aspects of good project management. If there are any differences, they are along the dimensions of *team and project size*, *time scale* (daily vs. project), and *organizational level* at which the practices are defined (developer vs. manager).

For example:

- XP practices are geared towards the level of the development team.
- Scrum practices are oriented more towards management. Scrum encourages you to organize and prioritize so the useful functionality gets delivered at a reliable rate. XP-like practices can help you to get there, but Scrum does not demand them.
- Lean Software Development works at the team and management level so that you can deliver useful software while delaying decisions as long as possible, thus allowing for feature evolution. Lean principles guide the developer activities and planning practices, but the details are up to the implementer.
- The Crystal methods are about communication, and prerequisite for any agile (or indeed, effective) development process.

These methods are all about feedback, prioritization, and value-delivery within a time constraint. The methods are complimentary; each method provides detailed guidance at various levels, but every project needs to work at *every* level. Al methods assume some basic level of "process" (for example, no agile method can work unless the team uses an appropriate SCM process). An agile method gives one a framework, but every framework has gaps that projects need to fill in using their skills and knowledge. The methods assume that the teams will follow appropriate good practices appropriate with the values of the method.

All methods are, in the end, about managing uncertainty. All projects are about delivering functionality with the provided constraints. Agile methods have much to teach all of us about how to deliver good software on an appropriate schedule. The fact that we have "methods" is an artifact of people taking comfort in a packaged solution.

Comparing Requirements Management: Prescriptive versus XP and Scrum

Steve Bannerman
Simplify and Oxford University
steve.bannerman@comlab.ox.ac.uk

September 29, 2003

1 Executive Summary

The following sections contrast how developers manage requirements in the context of predictive methods in general and two adaptive methods specifically: XP and Scrum. In addition, Scrum is split into two perspectives: outside a Sprint (Scrum/O) and inside a Sprint (Scrum/I).

In order to highlight the differences, a simplified set of requirements management activities are assumed:

- Identification - identify and name a requirement.
- Elaboration - elaborate the details associated with the requirement.
- Estimation - estimate how long it will take to implement the requirement.
- Allocation - allocate the requirement to a software release.

1.1 Who manages requirements?

Predictive Customers, Requirements Analysts, Designers.

XP Customers and Designers.

Scrum/O The Product Owner, the Team, and Anybody.

Scrum/I The Scrum Master and the Team.

1.2 How do they manage requirements?

Predictive Customers and Requirements Analysts identify and elaborate them. Requirements Analysts and Designers estimate them. Customers allocate them to a release.

XP Customers identify them. Designers estimate them. Customers and designers allocate them. Customers and designers elaborate them.

Scrum/O Anybody identifies them. The Team estimates them. The Product Owner allocates them.

Scrum/I The Scrum Master and the Team elaborate them.

1.3 What do they manage?

Predictive A set of persistent requirement elaborations (document, use cases, database).

XP A set of transient requirement placeholders (cards) and a set of persistent acceptance tests (code).

Scrum/O A set of persistent requirement placeholders (spreadsheet).

Scrum/I A set of persistent tasks (spreadsheet).

1.4 When do they elaborate requirements?

Predictive After allocation and prior to design, as a group.

XP During design, when it's time to implement an individual requirement.

Scrum/O Never.

Scrum/I Whenever makes sense.

1.5 Where do the requirements come from?

Predictive From Users to Requirements Analysts to Designers.

XP From Customers to Designers.

Scrum/O From Anybody to the Product Owner.

Scrum/I From the Team.

1.6 Why use this method?

Predictive Reduce costs by specifying a release and removing the most costly type of defects (assuming exponential defect removal model).

XP Reduce costs by deferring elaboration and specifying requirements in a reusable format, automated acceptance tests (assuming linear defect removal model).

Scrum Reduce costs by deferring elaboration of requirements. Reduce costs by doing only what is necessary for each Sprint (assuming a complex process which is undefined).

Lean Thinking shows that Agile Methods are Similar

Tom Poppendieck

Agile Methods are more similar than different. Lean thinking provides a conceptual framework which both rationalizes why agile methods work and provide guidance for tailoring them to specific situations. Lean thinking can be summarized in 7 principles:

Principle	Typical Application
1. Eliminate waste	• If an activity is not valued by the customer, don't do it. Eliminate waiting
2. Amplify learning	• Iterative development and incremental delivery. Concurrent Development of requirement details and code and tests.
3. Decide as late as possible	• Implement only what you need for the current step. Defer design commitments until you need them to make progress.
4. Deliver as fast as possible	• Use short iterations. Deliver working software every iteration, Highest value first.
5. Empower the team	• Focus on the people. Train them, then trust the team to decide how to do their work.
6. Build integrity in	• On-going detailed customer participation and customer tests. Test driven & domain driven development
7. See the whole	• Focus on results, not intermediate artifacts. Avoid sub-optimizing. Collaboration comes from measuring the team, not individuals.

These principles have been proven repeatedly in many disciplines to deliver larger economic returns, higher quality, better customer satisfaction, and better work-life experiences. The principles work for lean manufacturing, lean product development, lean logistics and they are now proving themselves in the domain of software development.

An important impact of this similarity of conceptual foundation is that any given team will need to tailor whatever method they start with to address the unique context of their project. All the agile methods advocate learning and adapting their practices. The principles of lean thinking provide guidance about HOW to tailor. Changes which are consistent with the principles, even if inconsistent with the particular agile method the team starts with are likely to lead to effective results.

Specific agile methods support tailoring in different ways. XP provides the four values: courage, simplicity, communication, and feedback. Scrum supports it by virtue of being underspecified. Crystal supports tailoring by permitting the team to do whatever it needs to in addition to a very small set of baseline practices. The lean principles support tailoring any method.