The Agile Times

A quarterly publication of the AgileAlliance

VOLUME V ISSUE 1

Inside this issue are 90 pages of extremely well written article. Once again, thanks to the community of practitioners, observers and writers.

--Ken Schwaber

UPCOMING AGILE CONFERENCES

XP2004, June 6-10, Garmisch-Partenkirchen, Germany www.xp2004.org

Canadian Agile Network Workshop, June 20-21, Banff, Canada www.agilenetwork.ca/ws2004

Agile Development Conference, June 23-26, Salt Lake CIty, Utah www.agiledevelopmentconference.com

XP/Agile Universe ,August 15-18, Calgary, Alberta, Canada www.xpuniverse.com

Scrum Gathering, October 13-15, Denver, Colorado

THE SCRUM GATHERING IN VIENNA

Boris Gloger

The Scrum Gathering 2004 in Vienna in April was a small step for each individual Scrum Master, but a big step for the evolution of Scrum. About 30 persons showed up with a wide range of knowledge about Scrum and team facilitation.

We not only had very good presentations from Ken Schwaber, Joseph Pelrine, Esther Derby, Diana Larsen, Norman Kerth, Gillian Attard and Boris Gloger, but we had a wide range of open discussions amongst the different groups: Scrum Virgins, Baby Scrumies, Scrum Experts, and Retrospective Facilitators. Scrum Virgins are people who had not had any involvement in Scrum, Baby Scrumies are just certificated Scrum Masters and Scrum Experts are the ones who already had expertise in using and implementing Scrum.

We did not make this distinction upfront; it was just a way of organizing people during the Scrum Gathering and was introduced by the attendees themselves.

For me, the Gathering got its value from the discussions across the different groups and from the fact that the Scrum ideas were able to influence the retrospective facilitators and vice versa.

I cannot go into all the details of the Gathering because it would take days, and I have a deadline to meet. But I will post the outcome of the Gathering within the next few days to the Scrum Alliance web site (http://www.scrumalliance.org/???) and I will publish a longer article about the Gathering in the next issue of The Agile Times. In case you want to sent me your impressions about the Gathering, please feel free to send me yourcomments to boris@scrumalliance.org.

Volume V; Issue 1

Topics

Editorial Staff

Marco Abis, abis@agilemovement.it, Agile Europe

Steve Berczuk, steve@berczuk.com, Agile Foundations

Scott Bogartz, scottbogartz@yahoo.com, Selling Agile To Management

Jeremy Brown, jeremy@quero.com, Book Corner

Chris Celsie, ccelsie@idirect.com, General Editorial

Mark Clifton, webmaster@knowledgeautomation.com, Unit Testing

Mike Cohn, mike@mountaingoatsoftware.com, Best Practices

Lisa Crispin, lisa_crispin2001@yahoo.com, Introducing Agile To New Environments

Esther Derby, derby@estherderby.com, Agile Project Management

Bryan Dollery, bryan@greenpulse.com, Agile Sociology And Psychology

Boris Gloger, boris.gloger@chello.at, Agile People and Sociology

Cliff Gregory, cliff@gregory.net, Agile Management

Mike Griffiths, mikeg@quadrus.com, DSDM

Michael Ivey, mdi@iveyandbrown.com, Scrum Success Stories

Martha Lindeman, mlindeman@agileinteractions.com, Agile Interactions

Brian Marick, barick@visibileworkings.com, Agile Testing

Trevor Mather, tmather@thoughworks.com, The Thoughtworks Perspective

Kent McDonald, kent@madsax.com, Agile Project Management

Raghu Misra, raghu@shipxpress.com, Agile Distributed Teams

Dan Pierce, dan@embeddedeng.com, Embedded Software

Mel Pullen, mel.pullen@symbian.com, Hard Questions For Hard Projects

Meade Rubenstein, Project Processes Tricks and Tips

Nancy Van Schooenderwoert, vanschoo@rcn.com, Ask the Experts

Andy Winskill, andy.winskill@rosewoodsoftware.com, The Agile Enterprise

Ken Schwaber, ken.schwaber@verizon.net, Editor in Chief

Carey Schwaber, Production Editor

Table of Contents

Agile Testing **Volume V, Issue 1**
Brian Marick - Editor

How We'd Do Testing on an Agile Project

Brian Marick (marick@testing.com)

If I were starting up an agile project, here is how I'd plan to do testing. (But this plan is a starting point, not the final answer.)

I assume the programmers will do test-driven design. That's well explained elsewhere (see the Further Reading), so I won't describe it here (much).

Test-driven programmers usually create their tests in what I call "technology-facing" language. That is, their tests talk about programmatic objects, not business concepts. They learn about those business concepts and business needs through conversation with a business expert.

Nothing will replace that conversation, but it's hard for programmers to learn everything they need through conversation, even the frequent conversations that a collocated business expert allows. Too often, the business expert is surprised by the result of a programming task - the programmer left out something that's "obvious". There's no way to eliminate surprises entirely - and agile projects are tailored to make it easy to correct mistakes - but it's sand in the gears of the project if, too often, a programmer's happy "I'm done with the order-taking task!" results in quick disappointment.

It's better if conversations can be conversations *about* something, about concrete examples. When those concrete examples are executable, we call them "tests" - specifically, I call them "business-facing" tests.

A business-facing test has to be something a business expert can talk about. Most business experts - not all - will find tests written in Java or C# too painful. A safe and satisfactory choice is to use Ward Cunningham's Fit (http://fit.c2.com). In it, tests are written as a variety of HTML tables that look not too dissimilar from spreadsheet tables. Fit is ideal for tests that are data-centric, where each test does the same kind of thing to different kinds of data.

Some tests are processing-centric, where each test is composed of a different set of processing steps. Tables are more awkward for that. I would augment the official version of Fit with my own StepFixture. It makes processing-centric tasks more compact. (See the end notes for its location; like Fit proper, it's open source.)

More important than the format of the tests is how they're created: collaboratively. The conversation begins with the business expert describing a new feature. This is often done in general terms, so it's important to train the team to say, "Can you give me an example of that?" whenever there's a hint of vagueness. Those concrete examples will help the programmers understand, and they may well also make the business expert suddenly call to mind previously overlooked business rules.

Those examples turn into business-facing tests, but I think it's important that they not *start* that way. I don't want to see people huddled around a screen, editing tables. It's too easy to get distracted by the tools and by making things tidy. I want to see people in front of a white board, scribbling examples there. Those examples can later be put into files.

What's the role of the tester in all this? One part, probably, is clerical. Guess who gets to turn scribbling into tables? But the more important parts are as translator and idea generator.

Experts are characteristically bad at explaining why they do what they do. Their knowledge is tacit, and it's hard for them to make it explicit. It's the tester's responsibility to draw them out. Fortunately, many testers are quick studies of a domain - they've had to be. It's also the testers' responsibility to think of important ideas the business experts and programmers might overlook. For example, both business experts and programmers tend to be focused on achieving return on investment, not on loss. So they concentrate more on what wonderful things a new feature could do, less on what it shouldn't do if people make mistakes (error handling) or intentionally misuse it (security). The tester should fill that gap, make sure the tests describe enough of the whole range of possible uses of the feature.

Quite likely, most of the tests will come after the programmer's started writing the feature. I'd want the initial set of tests to be enough for the programmer to estimate accurately enough and get started quickly. The tester can then produce additional tests in parallel. Always, any doubtful cases - "what *should* the program do here?" - will be reviewed by the business expert. As time goes on and the whole team learns the domain, fewer and fewer cases will be doubtful. The team will get better, in all ways, at making choices that make sense for the business.

The programmer will use the tests in something like the standard test-first way. When working on technology-facing tests, the programmer watches a new test fail, makes a small change to the code, watches the test now pass (and earlier ones continue to pass), cleans up if necessary, and repeats. The same thing will be done with business-facing tests. When I use Fit, I display its results in the browser. My cycle starts by looking at the first table element that isn't right (isn't green, in the case of checks; or white, in the case of actions). I flip back to my programming environment and do what's required to make that element right. If the job is simple, I can do it directly. If it's too big a job to bite off at once, I use technology-facing tests to break it into smaller steps. When I think I've got it right, three keystrokes run the tests, take me back to the browser, and refresh the page so I can see my progress. (It's a measure of the importance of rapid feedback that those three keystrokes feel like an annoying slowdown.)

All this is an important shift. These tests are not, to the programmer, mainly about finding bugs. They're mainly about guiding and pacing the act of programming, about making the evolution of the code toward a form that satisfies the business expert an evolution that's smooth and pleasant. Testers are not traditionally in the business of making programmer's lives pleasant, but the good tester on an agile team will strive to present the programmers with just the right sequence of tests to make programming smooth. It's too common for testers to overwhelm the programmers with a barrage of ideas to keep track of.

I'm edging here into the most controversial thing about testing on an agile project: testing is much more explicitly a service role. The tester serves the business expert and, especially, the programmers. Many testers are, to say the least, uncomfortable with this role. They are used to seeing themselves as independent judges. That will effect hiring: with exceptions, I'm more interested in a helpful attitude, conversational skills, and a novelty-seeking personality than testing skills. It's easier to grow the latter than the former.

By that token I am (with an important exception) content with having no testers on the project. If the programmers and business expert can and will develop testing skills, there may be no need for a person wearing a hat that says "Tester" on it. But, in addition to seeing programmers grow toward testers, I'd be happy to see testers grow toward programmers. I'd be ecstatic if someone I hired as a tester began to contribute more and more code to the code base. Agile is about team capability: as long as the team gets the work done, I don't care who does it.

Now for various exceptions.

Automated business-facing tests are the engine of my agile project, but I'm leery of automating everything. In particular, I will absolutely resist implementing those tests through the GUI. They are written in business terms, not GUI terms. They are about business value, not about buttons. It makes no sense to translate business terms into button presses, then have the GUI translate button presses into calls into the business logic. Instead, the tests should access the business logic directly, going "below" the GUI. The prevalence of automated GUI tests is a historical accident: testers used to *have* to test through the GUI because the programmers were not motivated to give them any other way to test. But agile programmers are, so our tests don't have to go through the traditional nonsense of trying (often fruitlessly) to make our tests immune to GUI changes while not making them immune to finding bugs.

But what about the GUI? How is it tested? First, realize that testing below the GUI will keep business logic out of the GUI. If there's less code there, less can go wrong. Moreover, a thin GUI offers less opportunity for "ripple effects" than does code deeply buried in the business logic. Selected technology-facing tests (for example, using jsunit for javascript input checking) plus manual tests of both the functionality and usability of GUI changes should suffice. We shouldn't need full automation.

I would expect arguments about this. I'd also expect to win them, at least at the start. But if my simple practice really did let too many bugs past, I'd relent.

Another manual testing practice is derived from my passion for concreteness. Few people buy cars without test driving them. That's because actually *using* something is different than reading about it or talking about it. Both of those acts are at a level of remove that loses information and narrows perception. An automated test is at the same level. So I would introduce a form of semi-structured manual exploratory testing into the project.

I'd do that by piggybacking on my fondness for end-of-iteration rituals. Some agile projects have a ritual of demonstrating an iteration's results to anyone they can drag into the project room. After that, it'd be natural for people to pair off (team members with team members, team members with observers) to try the product out in ways interesting to the business expert. One pair might look at what the web site feels like over dialup lines; another might emulate a rushed and harried customer-service person who's making mistakes right and left. They're looking not just for bugs but also especially for new ideas about the product - changes that can be made in later iterations that would make it better even though it's not wrong now. They're throwing up potential work for the business expert to decide about.

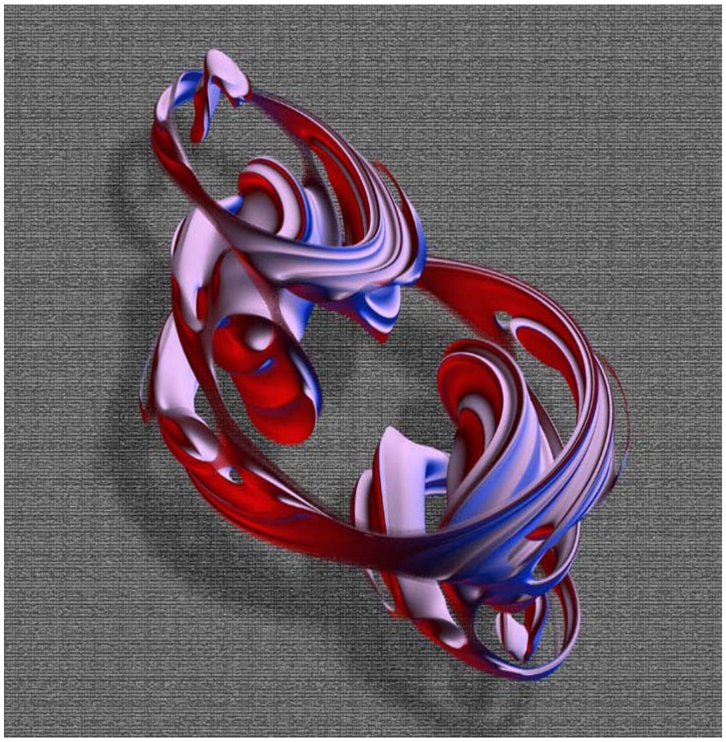
I would make a final exception for types of testing that are both highly specialized and don't lend themselves to a test-first style. Some of these types of testing are security testing, performance testing, stress/load testing, configuration testing, and usability testing. I don't now see any reason for these to be done differently on agile projects.

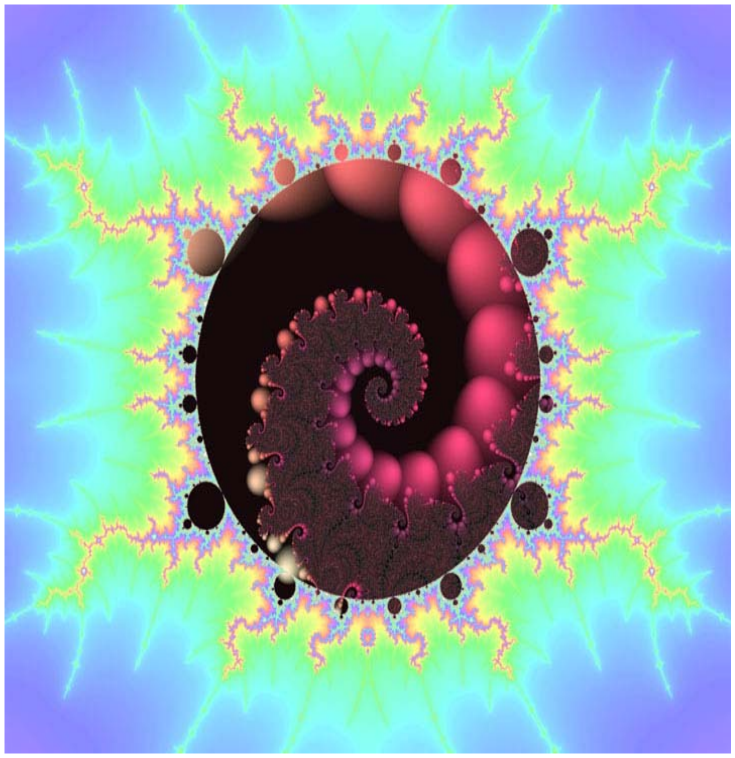
FURTHER READING

For test-driven design, I'd start with Kent Beck's *Test-Driven Development: By Example* and also read either David Astels' *Test-Driven Development: A Practical Guide* or Hunt and Thomas's *Pragmatic Unit Testing*. See also http://www.testdriven.com.

The best web site for exploratory testing is James Bach's: http://www.satisfice.com/articles.shtml. Kaner, Bach, and Pettichord's *Lessons Learned in Software Testing* is the closest thing to a book-length treatment. It's not very close, but it has a wealth of complementary material.

Fit is explained in Mugridge and Cunningham's *Getting Fit for Software Development* (forthcoming). My StepFixture is documented at http://www.testing.com/tools.html#StepFixture. The best web site for exploratory testing is James Bach's: http://www.satisfice.com/articles.shtml.





Volume V, Issue 1

Observations of Improvised Practices Used by XP Teams

Allen Higgins

OBSERVATION OF IMPROVISED PRACTICES USED BY XP TEAMS

OBSERVATION OF IMPROVISED PRACTICES

As the manager of a software development team I've been grappling with the challenges posed by adapting and developing management approaches for XP software development. The job of taking on an XP isn't made any easier by the need to "roll-your-own" implementation because the founders of XP provided a vision of the future, but cleverly left out the roadmap for how to get there. We focused on XP in 1999-2000, and were posed with the problem of adapting our organization, processes, procedures, even behaviours and modes of interpersonal interactions to support the change. Sure there are lots of resources out there on XP, but the XP community steers a little wide of prescription, particularly when it comes to organisational change and management strategies.

We were fortunate enough to be attempting to do this at the same time that a number of other groups we knew were attempting to do the same, so we had the added benefit of being able to look over each other's shoulders. An XPSIG was started up, meetings were hosted by various companies working in the space, and a lot of fruitful communication started flowing, and continues to flow, among the local development community. It's opinionated, fun, chaotic and very, very interesting.

The problem we've all faced is how to adapt the key practices (and values and variables) of XP to guide us in building best practice for our software development processes. It soon becomes obvious to anyone looking at XP that it implies or requires considerable embellishment and infrastructure to actually work. Not only are the key practices a pre-requisite, but the key practices imply the existence of a technological, social and cultural infrastructure which might be termed 'XP practice.' My view is that 'XP practice' installs the 'technology of XP,' without which XP is just a grab-bag of development practices.

What was really interesting about the XP implementations I saw around me was the number of big visible charts and wall diagrams that were used. One chart in particular seems to capture and distil a lot of the key practices of XP into a single focal point. It's more of a system than an single chart; a status board acting as the catalyst for volunteering, stand-up meetings, stories, communicating, planning, reviewing and a load of other stuff. For the purpose of this discussion let's call it the 'task-board process.'

A BEST PRACTICE TASK-BOARD PROCESS?

Because we shared our ideas and experiences through the XPSIG, in direct conversation and by visiting each others sites, the 'task-board process' was something that all the teams had experimented with. Some of the teams claimed it was a key success factor in their adoption and use of XP, in others this aspect failed.

The 'task-board process' could be described as the combination of regular stand-up meetings, story-cards and a cork board. It appeared to act as a powerful enabler for many other elements of XP. The teams using the process continue to evolve the format and the aspects of XP, process, and management practice it incorporates such as stand-up meetings, task management, volunteering, estimation, and planning. For the teams that continue to use the process, it operates as a focal point for them and for their interpretation of what XP is. The 'task-board process' is explained as offering a systematic framework for aspects of XP practice, more mundane perhaps than 'pair programming' but it can be more easily institutionalised or appropriated by an organisation than some of the more radical aspects of XP.

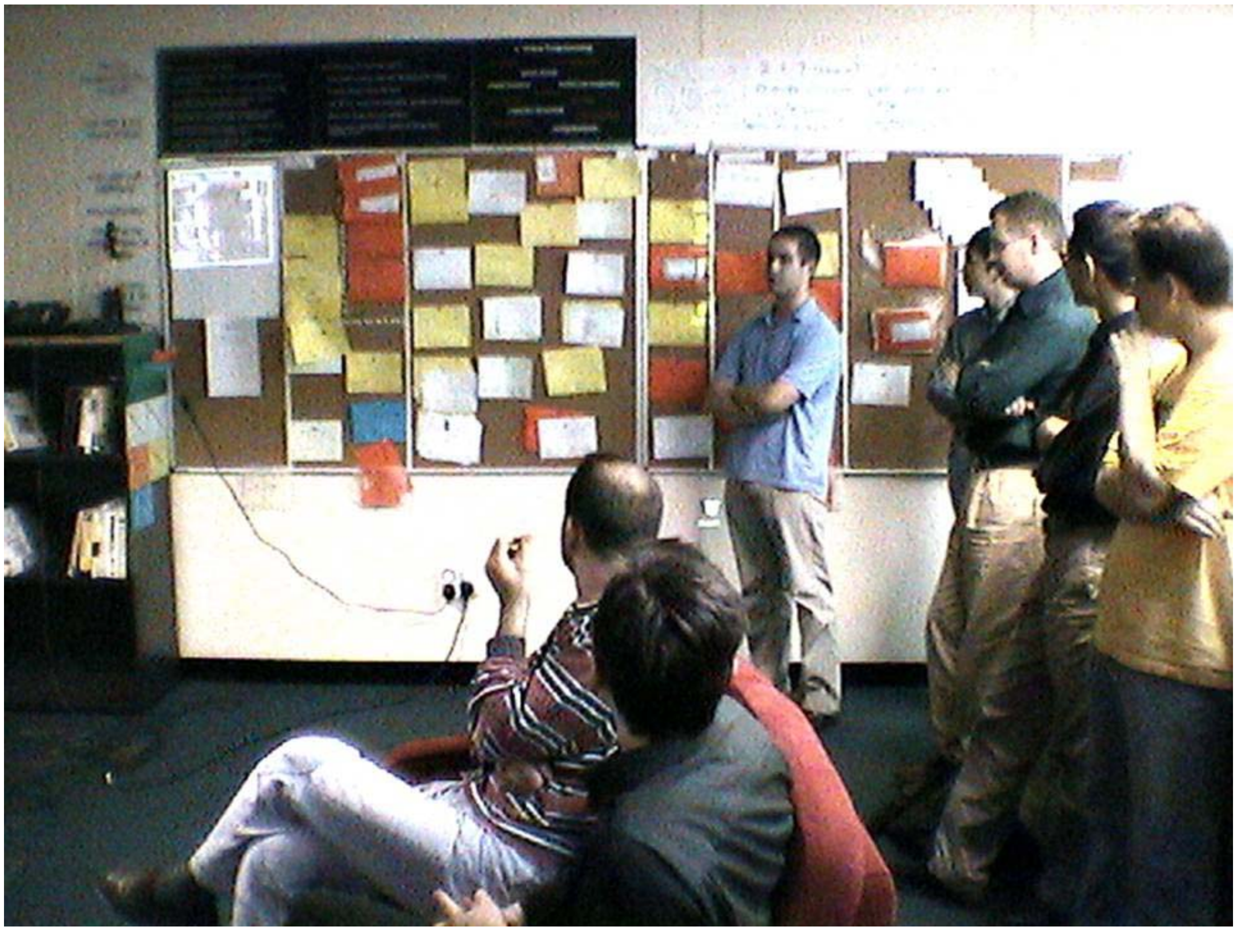


Figure 1 A team holding a daily stand-up meeting in front of the task-board

The typical 'task-board process' occurs around a short (20 minutes max), regular (often daily) stand-up meeting of all team members (developers, customers and anyone else interested). The meeting has some rules: it takes place in front of a physical task-board, and everyone takes turns to talk briefly about their current story (progress, problems, plan or cry for help). The story must be represented by an actual story-card (no invisible jobs). In fact the person is expected to walk up to the board and point to the cards referred to. Separate break-out meetings frequently follow on after the stand-up. The task-board itself is typically organised along the following lines (see figure 2). Stories are added in queues at one end, generally in order of priority (highest at top) and they are volunteered or 'signed-up' for by engineers when they have completed their current story-card. It is generally considered bad form for a developer to have a large stack of in-progress story cards.

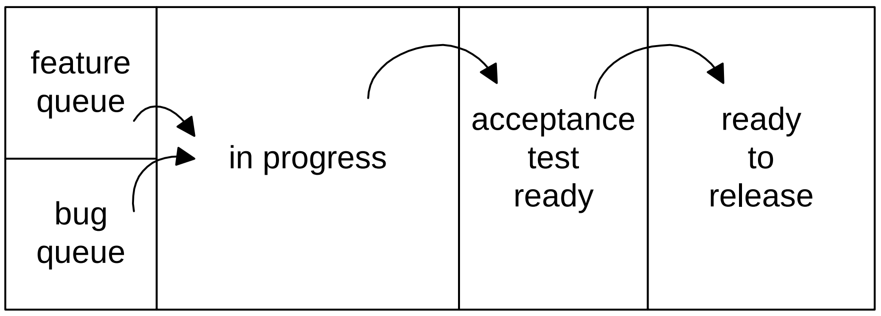


Figure 2 Flow of story-cards on the task-board

When a story-card is adopted by a developer, it is moved into the 'in progress' area and is then talked about at the regular stand-up meeting. The act of talking about the story-card is given as one of the really useful reasons for holding these sessions regularly (daily) as they serve the dual function of sharing information and alerting other engineers to a request to pair program or for assistance.

Those teams still using the 'task-board process' appear to have had considerable initial success with it. Comments from developers are typified by this example:

"What I think about the task-board, and I feel it myself, is that the engineers have a hell of a lot more autonomy now. In what they do, there is much less control about what we do now, we pick things off the board, ourselves and we drive them ourselves right through to the end"

As a 'Best Practice' for XP and software development it appears that something like the 'task-board process' supports responsible self selection for tasks (volunteering for stories) rather than resorting to direct assignment from supervisors. Manager comments on the 'task-board-process' tend to make supportive comments on this change;

"...if some people have some spare time they will take up things themselves, they do volunteer for it"; "...in a way it's empowering the engineers, they're controlling their own work rate, their own deliverables."

These claims however are tempered somewhat by statements from some developers along the lines of:

"It's a combination of self selection and being 'volunteered'"

Notably a manager in a team that gave up using the 'task-board process' stated:

"No one volunteers, they know who should do what, and they just get on with it"

ACCESS TO INFORMATION

It appears that something like the 'task-board process' has a role in teams experimenting with XP as a means of embedding many of the key practices of XP into the fabric of the organisation, for example by replacing direct supervision with transparent systemic control arising from the presence of the group of developers and other contributors during the stand-up meeting. Although not elevated to the level of a key practice in the literature on XP, I think the 'task-board process' is notable in that it has operated successfully in several groups adopting XP as their core development process. Comments on the practice by members of teams where it was successful includes language like 'benefit', 'responsibility', 'ownership', 'visibility' and 'control'. Perhaps it is the roll-your-own nature of aspects of XP practice that affords XP the opportunity to be owned or deeply appropriated by those organisations using it.

Allen Higgins

These adaptations and improvisations by participants in the XP teams I saw appear to represent a conscious attempt to free up the control of information in the organisation. This potential carries significant risks of failure as it characterises a shift of power or control between and within teams and organisations. However home-brewed processes like the 'task-board process' offer the possibility of greater transparency and access for tracking the activities and progress of development for all, thereby reducing the uncertainty surrounding development and dependent activities. This in turn should allow contributors and participants to cope better with the creative and uncertain process of development and serve to support a dramatic improvement in capability of an XP enabled development team.

March 2004

Department of Management Information Systems

The Michael Smurfit Graduate School of Business

National University of Ireland

University College Dublin

Product Review : VersionOne

John Brothers

OVERVIEW

VersionOne (www.versionone.net) is a commercial software tool for managing agile projects. My company has been using it for about 6 months to manage project delivery using the Scrum (www.controlchaos.com) methodology. Previous to V1, as we started to introduce Scrum, we were using a combination of Twiki pages and Excel spreadsheets to manage the projects, and we found that to be cumbersome to use, and tedious to maintain. So when I heard about VersionOne, I called them, and arranged for a demo.

INSTALLATION

Unlike many of the alternate tools, VersionOne is written in C# and ASP.Net, and runs only on Windows servers, with SQLServer as the database. Luckily for us, this didn't present a problem as our administrative systems are all Windows-based. Once we had that server up and running, installation was about a 10 minute process (although I had a salesman from the company installing it) for me.

SETUP

The first thing I had to do was create users – it comes with a default Admin user, and you use him to create the rest of the users. Creating users was no problem, and I could set them up as *admin*, *members* or *viewers*. You can't see anything when you don't have a login, so everyone who may want to look at or participate needs at least a *viewer* level access. And then it was on to creating projects, releases and sprints.

USER REQUIREMENTS

VersionOne uses a number of fancy IE tricks to make the user interface easier and more accessible. These, of course, don't work in non-Windows environments, much to the chagrin of our dedicated Linux and Mozilla users (such as the author). I do have to admit, however, that in at least one area—the Sprint Planning page—the IE tricks are pretty cool.

USAGE

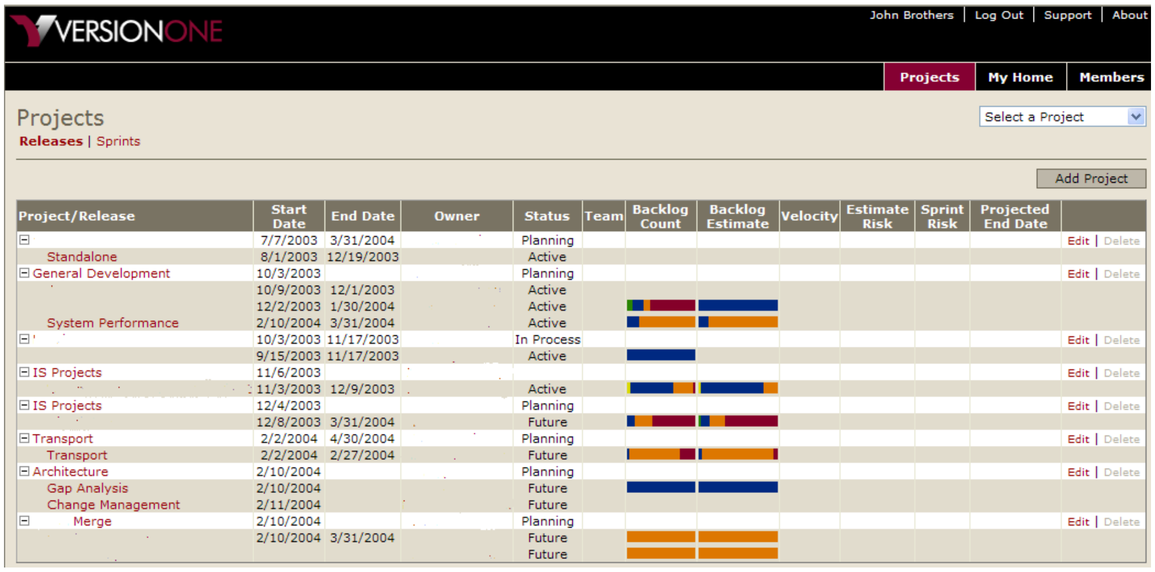
There are actually three different flavors of VersionOne, focused on XP, Scrum and DSDM-style project management. I specifically chose the Scrum interface, but I found that there were a number of configurable features and options that I could turn on or off to tweak the behavior to fit my company's needs. Since we aren't an XP shop, XPlanner didn't seem like an appropriate tool for us, so I don't have any compare and contrast examples. VersionOne does come with a clear and well-written user's guide, which I found quite helpful early on.

VersionOne for Scrum generally uses the following hierarchy:

Projects
- ➔ Releases
  - → Sprints
    - → Backlog Items
      - → Tasks

However you can also have Sprints that are not inside Releases (just manage them within the project) and you can have Backlog items in Releases as well. This makes for a very flexible approach, which can sometimes be confusing. It does conform well to the Scrum model – with a "general" backlog of items that aren't assigned to any sprint (yet) and a set of backlog items assigned to the current working Sprint. V1 has a number of simple tools to make it easier to create Sprints semi-automatically, easily populate them with Backlog Items and then maintain the Sprint Backlog over the course of the project.

Here is a screenshot of our current project list, with the names blanked out to protect the guilty:



Projects
Releases | Sprints

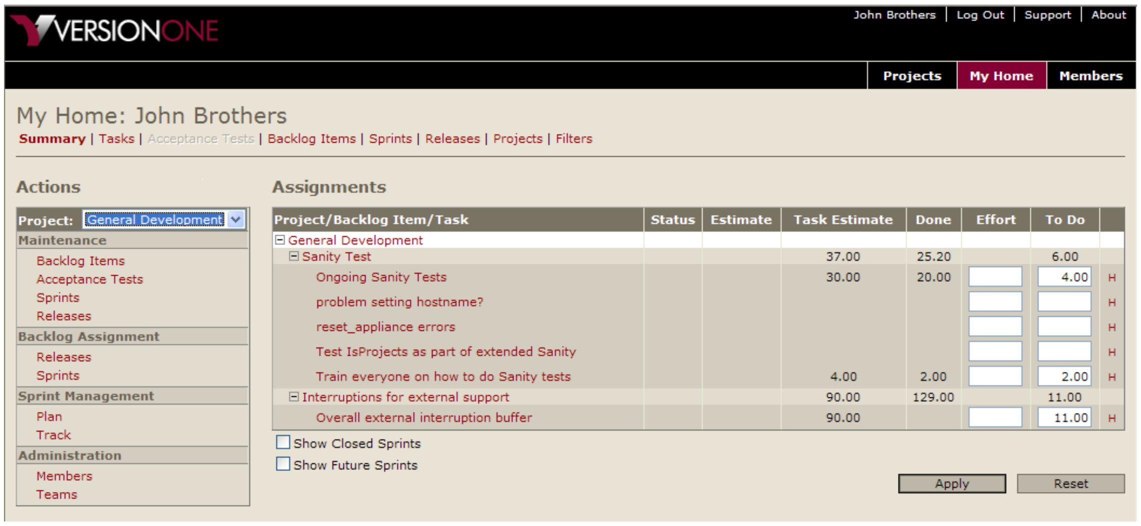
Project/Release	Start Date	End Date	Owner	Status	Team	Backlog Count	Backlog Estimate	Velocity	Estimate Risk	Sprint Risk	Projected End Date	
⊟	7/7/2003	3/31/2004		Planning								Edit | Delete
Standalone	8/1/2003	12/19/2003		Active								
⊟ General Development	10/3/2003			Planning								Edit | Delete
	10/9/2003	12/1/2003		Active								
	12/2/2003	1/30/2004		Active								
System Performance	2/10/2004	3/31/2004		Active								
⊟	10/3/2003	11/17/2003		In Process								Edit | Delete
	9/15/2003	11/17/2003		Active								
⊟ IS Projects	11/6/2003											Edit | Delete
	11/3/2003	12/9/2003		Active								
⊟ IS Projects	12/4/2003			Planning								Edit | Delete
	12/8/2003	3/31/2004		Future								
⊟ Transport	2/2/2004	4/30/2004		Planning								Edit | Delete
Transport	2/2/2004	2/27/2004		Future								
⊟ Architecture	2/10/2004			Planning								Edit | Delete
Gap Analysis	2/10/2004			Future								
Change Management	2/11/2004			Future								
⊟ Merge	2/10/2004			Planning								Edit | Delete
	2/10/2004	3/31/2004		Future								
				Future								

The major items on the left are Projects, and the shaded elements underneath them are Releases. We rigorously associate Sprints with Releases, so you don't see them here, but when you click on a particular Release, the list of Sprints pops up. Note that we track actual time spent as well as estimates, which is not mandated by the Scrum methodology, but which VersionOne handles just fine.

The multicolored bars in the middle represent quick status for the Backlog – future, in development, complete, etc – both in terms of raw numbers of items and in Estimated time remaining. One thing I like about V1 is that it tries very hard not to associate a meaning to these estimates – I think of estimates in hours, but it doesn't make that distinction, so an estimate of 10 might mean hours, days, weeks or jelly beans.

We don't create new projects every day, so most of our work is done at the Task and Backlog item level. And V1 handles the creation and maintenance of Tasks and Backlog Items quite well, although we found the UI to be somewhat difficult at first (but that definitely got easier with more use).

Once the team members were assigned to the project and given tasks, the "My Home" page became very valuable.



My Home: John Brothers

Assignments

Project/Backlog Item/Task	Status	Estimate	Task Estimate	Done	Effort	To Do	
General Development							
Sanity Test			37.00	25.20		6.00	
Ongoing Sanity Tests			30.00	20.00		4.00	H
problem setting hostname?							H
reset_appliance errors							H
Test IsProjects as part of extended Sanity							H
Train everyone on how to do Sanity tests			4.00	2.00		2.00	H
Interruptions for external support			90.00	129.00		11.00	
Overall external interruption buffer			90.00			11.00	H

This is my personal page, representing the various tasks that are assigned to me. It gives me a very quick way to find out what I need to do for the project(s) I'm on. As you can see, I can instantly update the amount of work I've done and the amount of work left to do on this page.

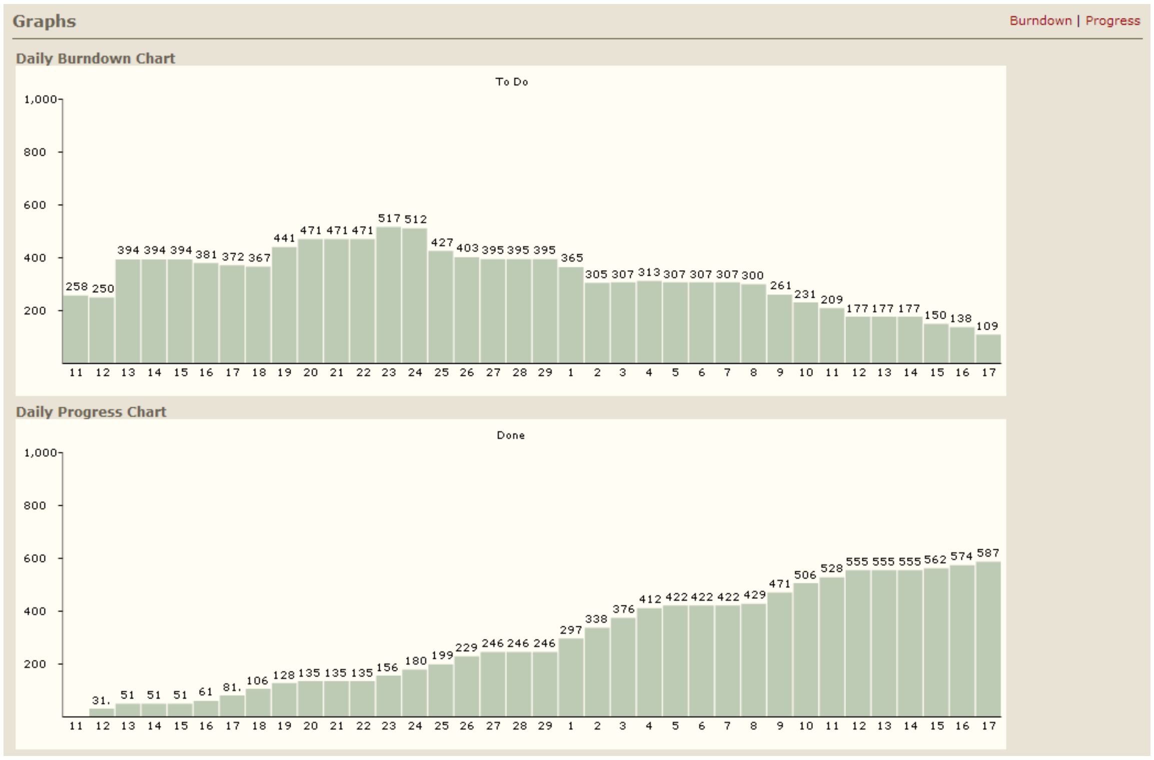
Configuration

One of the areas that V1 really shines is in its ability to be configured – you can add categories, priorities, statuses, etc to your hearts content to the existing lists, and define the order. V1 doesn't know what they mean, but it knows how to distinguish them, and displays that with a simple graphic (as shown in the project picture above).

In addition, V1 handles incompleteness very well – don't have any tasks for a backlog item? No problem. Don't know what the status is, don't want to make an estimate yet? No problem there either. Very few fields are required to create Backlog or Tasks, and any of those fields can be revisited at any time.

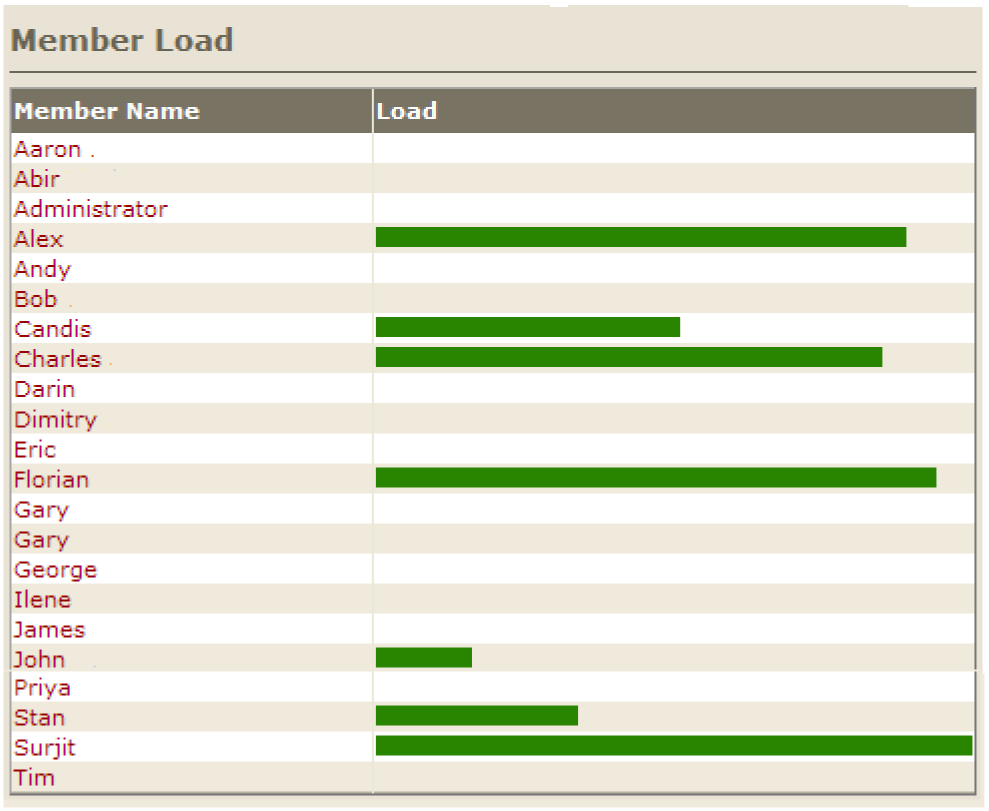
TRACKING/REPORTING

Scrum's concept of the Burndown chart as a key indicator of project health is very well supported by VersionOne. It automatically generates and maintains the graph for you, scaling them appropriately, and they're easy to read and follow



Member Load

V1 also handles member load – once estimates are put in, it keeps very accurate track of the number of "whatevers" (hours, gummi bears, etc) associated with the estimates, and can differentiate between work complete and work still outstanding.



Member Load

Member Name	Load
Aaron .	
Abir	
Administrator	
Alex	🟩
Andy	
Bob .	
Candis	🟩
Charles .	🟩
Darin	
Dimitry	
Eric	
Florian	🟩
Gary	
Gary	
George	
Ilene	
James	
John	🟩
Priya	
Stan	🟩
Surjit	🟩
Tim	

Change History

Change history is something that most traditional project managers care deeply about, and V1 is up to the task – every change to any field anywhere is logged – when it happened, who made the change, and what the value was before and after.

AREAS FOR IMPROVEMENT

User Interface

Especially at first, the UI can be confusing – there are often two links on a given page with roughly the same name, and they have different results. Luckily, this gets better after only a little practice, but it is still a battle to fight with the team early on. I'm told that many people find the interface quite intuitive from the beginning, but that wasn't our experience. VersionOne did win a Jolt Award, partially based on its usability relative to the other products in this field.

Support for Netscape/Mozilla

Even if we can't use the cool IE tricks, it would be nice if there was some sort of alternate way to work on the same pages. Right now, only a few pages work well with Netscape, not enough to make it usable on a day-to-day basis.

Risk Census

After reading "Waltzing with Bears" (DeMarco and Lister) I became very intrigued about the idea of building a Risk Census for the project, and since VersionOne didn't have that capability, I had to use our Twiki to manage that. I believe that the RiskCensus is a great bridge between traditional project management and agile project management, and it is my hope that V1 and other tools will incorporate this feature set.

Email Alerts

It would be great if V1 could be configured to email the team members every morning, reminding them to update their time. It would probably be even better if individuals were notified when new tasks had been assigned to them.

Loose Ends

It's not always obvious from the Backlog view if every task has an estimate and has been assigned to someone. It would be nice to have some sort of "audit" against a Sprint to determine which tasks had no estimate data, or weren't assigned to someone. This might not be valuable for everyone, but it would certainly be valuable for us.

Searching

You can't search for a Task by ID, which can make it hard to quickly specify "Go work on Task X". Searching for Backlog items is cumbersome – you have to go to the "Backlog Planning" page, and enter your search criteria there.

CONCLUSIONS

We use a separate system for time tracking, and at first it was difficult to get everyone to update their hours in both systems. However, as the team has gotten more used to V1, they have gotten better about updating their hours. And when they forget, a quick glance at the burndown chart shows the lack of progress, and a reminder email quickly goes out.

Since the last project I worked on has wound down, two additional projects have started up using V1. These latest projects are interesting because they include a mix of internal and offshore staff, and we're going to find out how well our agility works in that kind of environment.

And, very smartly, the V1 guys have set up an XP demo system right on their site that you can use and experiment with. So if you want to know more about what it can do, go to their website (www.versionone.net) and try it for yourself.

BIO

John lives in Atlanta with his wife, 3 kids, 2 dogs and 2 cats. He has 10 years experience building large-scale systems. He learned about agile development in 2000 and hasn't looked back since.

JUST-IN-TIME REQUIREMENTS ANALYSIS
AN AGILE APPROACH TO AN AGE-OLD PROBLEM

By Michael Lee

> Software development is a (*resource-limited*) cooperative game of invention and communication. The primary goal of the game is to deliver useful, working software. The secondary goal, the residue of the game, is to set up for the next game. [1] Dr. Alistair Cockburn, (emphasis added)

1 INTRODUCTION

In a world of limitless resources, the decision to build a piece of software would be driven entirely by need and requirements analysis would not be a problem. Business analysts would keep refining and project teams would keep developing until the needs were met.

In the real world however, the decision to build software is primarily an economic decision – with need only a secondary consideration. At some level, those purchasing software "on spec" (i.e. buying software before it is built) need to know ***beforehand*** what is being purchased and how much it will cost so an economic determination can be made – "is it worth it?"

Balancing economic considerations with the necessity to identify and communicate need requires tradeoffs on how we approach analysis and how requirements are identified, defined, and communicated.

Conflicting Needs

All projects share the same basic goals:

- Maximize value. Provide the most capabilities for the least cost.

- Reduce Risk. Reduce or eliminate the impact of project failures.

- Ensure Quality. Guarantee that the delivered system meets the expectations of all stakeholders.

Likewise, all requirements analysis efforts share, to a certain degree, a common set of perceived needs that must be addressed:

- Scope. What are the boundaries of the effort? Where are the boundaries flexible? Where are they fixed?

- Known Cost and Schedule. How much will it cost? How long will it take?

- Value. What are the benefits? Given the cost and schedule, are the resulting benefits worth the effort? As things change, how can value be maintained?

- Communication. Are needs and desires clearly, effectively, and unambiguously communicated? How are differing expectations handled (between business and developers; between purchasers and users; between customer and business)?

· Specification. Provide a blueprint that defines the expected behavior of the system. A blueprint used by developers to build the system and by testers to ensure the quality of the system.

· Planning. What needs to happen? When? What resources are required given the size and scope? What are the hard milestones that are driving the effort?

· Certainty. Is every important thing known? How are unknowns dealt with? How is chaos managed and its impact limited?

· Flexibility. How can the project adapt to changing business needs? New and evolving technology? How is the impact of change limited?

Unfortunately, these needs often conflict with each other and also with the overall goals of the project. Over time, analysis approaches have evolved to favor one or more items over others.

Traditional Analysis

Traditional Analysis tends to favor *known cost and schedule* and the *reduction of risk* over all other objectives. Decision makers and stakeholders want to know before they commit to a project what the costs are and when the software will be delivered. This increases the importance of *specification, known cost and schedule* and *certainty* at the expense of *communication, flexibility,* and *maximization of value* (figure 1)

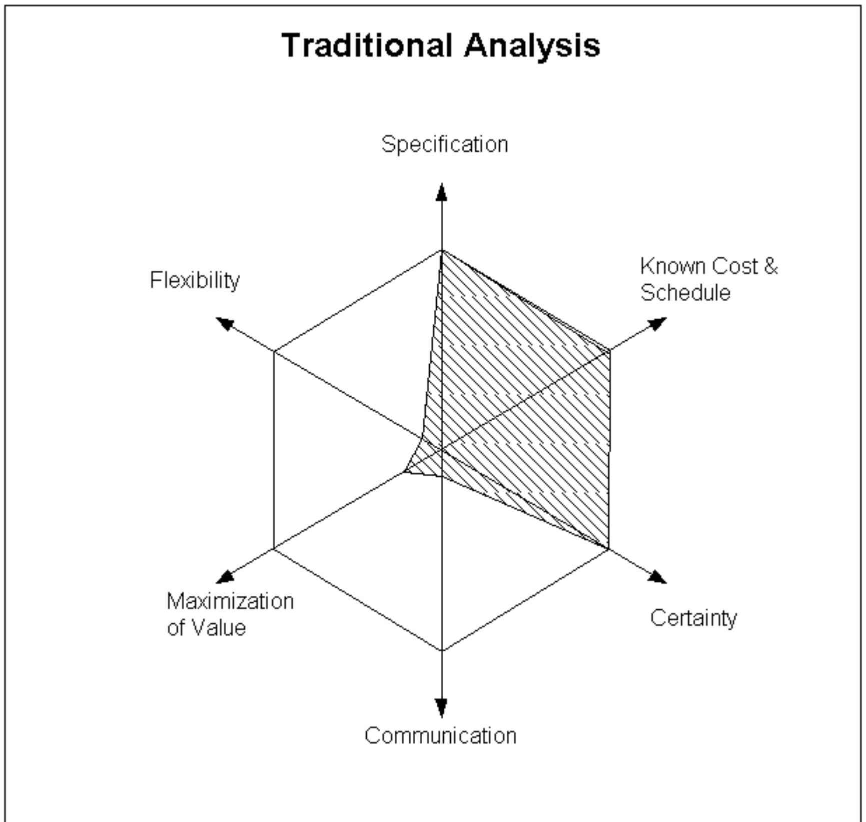


Figure 1

The approach taken in traditional analysis, along with the artifacts of the process, is indicative of what is important. Significant analysis time is dedicated upfront to ensure all the relevant details have been addressed in the cost and schedule. In an attempt to reduce risk, monolithic (and often massive) requirements specifications are produced and elaborate change management processes are put in place. The result is a process that is the antithesis of agile and one that often works at cross-purposes with its intended objectives [2].

The shortcomings of traditional analysis are obvious to even the most casual observer.

- Time dedicated to upfront analysis is time not spent developing solutions. Too much upfront analysis adds unnecessary cost and decreases the overall value of the delivered software [3] (figure 2).

- “Software requirements are a communication problem” [4] not a specification problem, and monolithic requirement specifications inhibit rather than enhance communication. Effective communication requires collaboration between participants. Requirements specifications rarely, if ever, facilitate collaboration.

- Static requirements increase, not decrease risk. [5]

· Change management processes don't protect a project against change; they only make change more costly.

· No amount of planning can eliminate all of the unknowns. Fixed schedules and static requirements inhibit a projects flexibility to react to changing situations.

· "No plan survives contact with the enemy." [6] No matter how much time upfront is spent on planning, something always changes. And the further out the planning goes, the more likely the plan will change.

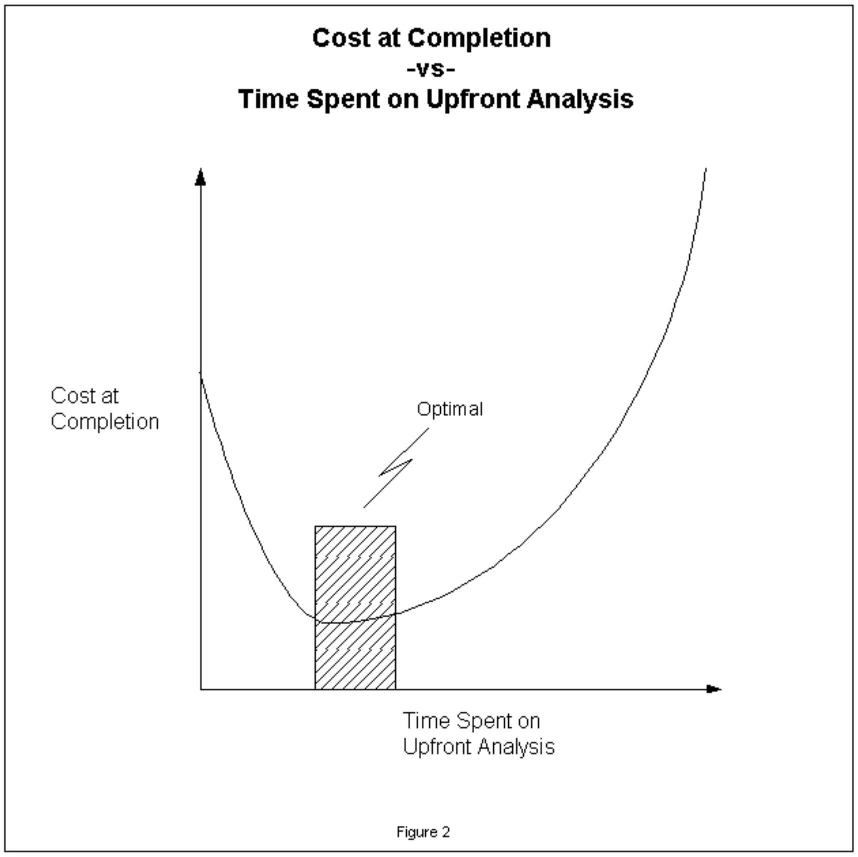


Figure 2

Story-Centric Analysis

On the opposite side of the spectrum from Traditional Analysis is Story-Centric Analysis. Story-centric Analysis tends to favor *maximation of value, communication,* and *flexibility* over all other objectives. A good example of Story-centric Analysis is found in Mike Cohn's book, *User Stories Applied*. [7]

With Story-centric Analysis, decision makers and stakeholders want to ensure the most capability is provided for the least cost in the quickest possible time and with the highest quality. This increases the importance of *communication, value, quality,* and *flexibility* at the expense of *known cost and schedule, certainty,* and *scope* (figure 3)

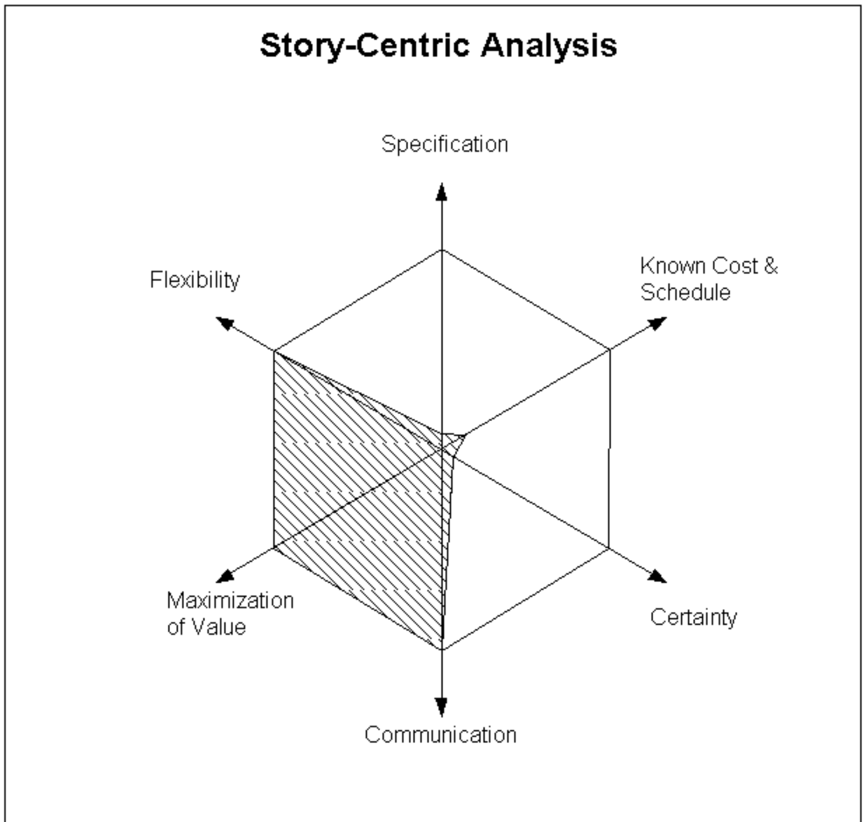


Figure 3

Overall, Story-centric Analysis is significantly better (if not overwhelmingly better) than Traditional Analysis and is a near-perfect approach for small-to-medium projects. However, when it comes to large-scale, (potentially) expensive, resource-intensive development efforts, Story-centric Analysis has its limitations.

· Scope tends to be fluid at best and ambiguous at worst. A great deal of attention is given to the parts – user stories and epics – and very little to the whole.

· It doesn't scale well. Story-centric analysis requires close collaboration within a tight-knit team. The larger the project – in terms of functionality and the number of people involved – the more likely it is that communication will breakdown at organizational boundaries.

· Cost and schedule are ambiguous. When a development effort is tasked with providing the most capability for a fixed price, this is not a problem. But in those situations where one has to deliver a fixed set of features for a specified cost, Story-centric Analysis is ineffective.

· Depth, but little breadth, of detail. One of the purposes of analysis is to gain an understanding of the problem to increase the level of certainty that all relevant areas have been (or will be) addressed. The focus on epics and user stories tends to narrow the focus of analysis and may result in important things being left out.

Just-In-Time Requirements Analysis

Just-in-Time Requirements Analysis (JITRA) is an agile approach to requirements analysis specifically tailored for large, (potentially) expensive, resource-intensive, enterprise-class development efforts. It combines the best features of Story-centric analysis with the familiarity of Traditional Analysis. Its primary focus is on *maximization of value, communication,* and *flexibility* while at the same time recognizing and addressing the needs of decisions makers on large projects (*known cost and schedule, certainty,* and *scope*) (figure 4) The remainder of this article details the principles of JITRA; outlines the JITRA process; and highlights how JITRA drives agile development processes.

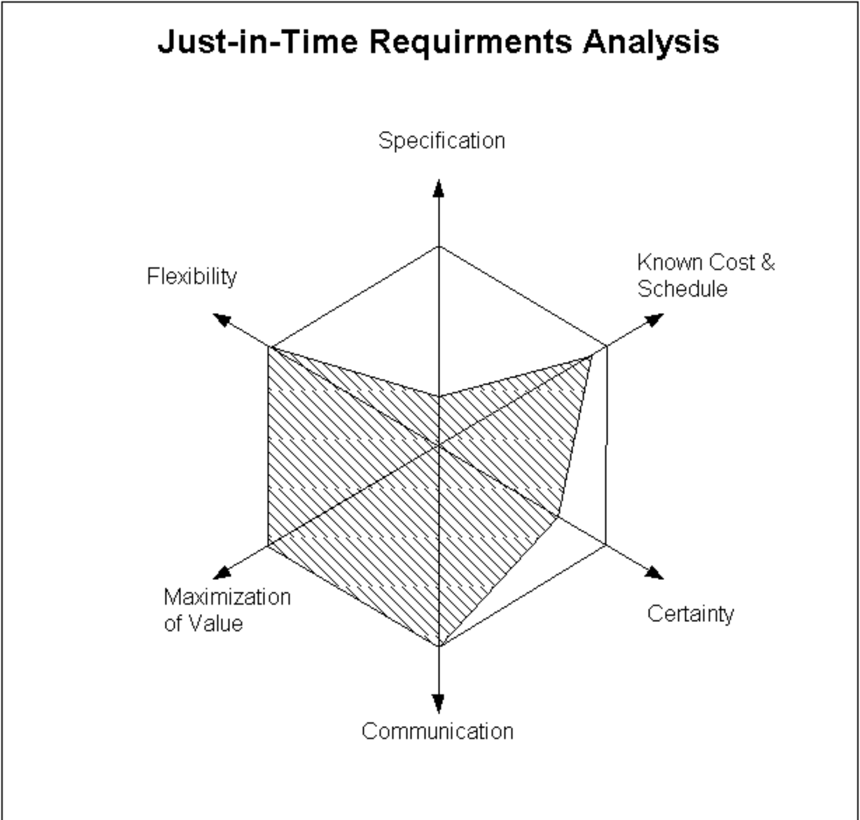


Figure 4

2 JITRA – WHAT IS IT?

Just-In-Time Requirements Analysis is an agile process based on the following principles:

- Last Responsible Moment. This is the fundamental principle of JITRA (the "Just-In-Time" part of Just-In-Time Requirements Analysis). Detailed analysis is deferred until the last responsible moment before requirements are needed.

- First-Things-First. Analysis is always focused on the most important things (as defined by the business, not developers). This requires the business to constantly assess and prioritize their needs.

- Understand the Problem. Analysis is not just about identifying and communicating requirements (i.e. need). It is an activity that allows organizations to gain a better understanding of the problem. This understanding can then be used to drive economic decisions.

- Collaborative Communication. Constant and effective communication between all participants in the analysis process is *critical*. Use all means available to communicate clearly and effectively and adapt the means of communication to fit the participants.

- Equality of Value. Once a project cost has been fixed, Equality of Value allows decision makers to adapt or change functionality while managing to the same fixed price. This is a key principle of JITRA and meets the critical need of decision makers to predict and manage costs - while at the same time preserving flexibility and agility.

- Concurrent Engineering. With the possible exception of some initial analysis, all analysis activities for follow-on development should be occurring in parallel with current development. This compacts the life cycle and increases value at delivery (although it may introduced added risk).

- Maximization of Value. Analyze value as well as functionality. Always examine the trade-off between need and benefit while focusing on the maximization of value.

- Focus on (Early) Delivery. The goal of analysis is delivery of software, not requirements specifications. In the history of software development there has never been a single requirements specification that has generated revenue for the organization purchasing the software[*].

- Learn By Doing. Analysis needs to be an active process. One of the objectives of analysis is to learn more about a problem, and often the best way to learn is by doing.

- Continuous Analysis. JITRA is primarily a continuous process of feedback and refinement. What we know tomorrow will always be more than what we know today. New knowledge needs to be continuously fed back into the analysis process and requirements need to be continuously analyzed and refined based on the new knowledge.

- Continuous Planning. Prioritized requirements drive planning. Since requirements and businesses priorities are constantly changing, planning must be continuous.

- Embrace Change. The JITRA process facilitates and embraces change. Instead of relying on elaborate change management procedures, JITRA allows (and in some cases encourages) changes to occur at anytime. Changes are simply treated as requirements that must be analyzed and prioritized as part of a continuous analysis process.

- Isolation of Complexity. Complexity adds cost, decreases value and makes things harder to change. Don't make the problem more complex that it needs to be. But where complexity is required, isolate it.

- Constant and Continuous Improvement (Kaizen). As new knowledge is fed back into continuous analysis, always look to improve on existing requirements. Refactoring applies to requirements just as much as it does to code.

3 THE JITRA PROCESS

Just-in-Time Requirements Analysis is a simple process. Analysis starts at the highest levels of abstraction and requirements are continuously refined over the life cycle of the project. The most important things are analyzed first, and the analysis is always at the level required to meet the current needs of the project – no more and no less.

To support this process, JITRA defines four major analysis activities:

- Initial Analysis
- Feature Set Analysis
- Detailed Story Analysis
- After-action Analysis

These activities are performed continuously throughout a project's life cycle, and may overlap each other in scope and detail[*]. The following sections discuss each of these activities in detail.

Initial Analysis

The Initial Analysis activity is performed at the start of a new unit of work[#]. This unit may be an individual system, a system of systems, a subordinate subsystem, or individual subsystem components. The purpose of this activity is to broadly define the scope for the work ahead; specify an initial set of features, functions, and capabilities required for the specified unit of work; and estimate cost. The specific tasks of Initial Analysis include:

- Defining scope. The first task of Initial Analysis is to broadly define the scope of the work ahead. This allows planners to estimate the level of effort required to complete the work. On some projects this take may take a few hours or days. On other projects – especially on those projects where a detailed estimate is required (i.e. fixed price projects) – this may take longer.

- Gain the required understanding. The second task of Initial Analysis is to gain a better understanding of the current problem domain. This doesn't have to be a detailed understanding, but as a minimum the team must have a reasonable expectation of success if the project moves forward.

- Develop the initial *global feature set.* The key task of Initial Analysis is to define a generalized

list of things the new system needs to do. This list might be a simple bullet list, or it might be detailed in one or more high-level stories.
The initial list does not have to be a complete list. The only requirement is that it contains enough information to begin follow-on analysis and development activities.

· Estimate the cost of the effort. Cost drives decisions. One of the most critical tasks of Initial Analysis is to estimate the cost of the overall effort. The quality of this estimate is a function of the time spent defining scope and defining the global feature set. Use caution, however. No estimate is perfect. No matter how much time is spent estimating cost, there will always be errors – "noise in the system – that cannot be eliminated.

· Assess the value of the global feature set. Evaluate items in the global feature set against their associated cost. Identify benefits and perform a Cost-Benefit Analysis to determine if the benefits are worth the price.

· Define a high-level schedule for the project. Identify a justifiable (and reasonable) delivery date for the first unit of work. Define a high-level roadmap for this delivery.

Scaling Initial Analysis

Initial Analysis is highly scalable and works extremely well on complex projects – especially those involving multiple teams and organizations.

The key to this scalability is the recursive definition of the unit of work in Initial Analysis. At the top-level, a unit of work may specify a system of systems. Initial Analysis is performed at the top level and then the unit of work is partitioned into one or more subordinate units of work (individual systems).

This hierarchical decomposition is recursive and may continue down any number of levels. At each level an initial analysis is performed and the unit of work is either partitioned into subordinate units of work, or Initial Analysis is completed and Feature Set Analysis is begun.

Participants in Initial Analysis

Analysis requires the -active- participation of every interested party. As a minimum, this includes:

· Stakeholders. The Requirements Stakeholders (decision makers, purchasers, customers, sponsors, users, etc.) represent the business. They provide input into the analysis process and make all of the business decisions related to Initial Analysis.

· Domain Experts. The Domain Experts are the people who best understand the problem – and how to solve it. They typically include expert users of existing systems and experts in a company's business processes.

· Business Analysts. This is probably the most misunderstood role on the entire team - the direct result of existing perceptions. Unlike most existing views of Business Analysts, this role does not define a gatekeeper or intermediary. In JITRA, the business analyst is the expert problem solver who performs the bulk of the analytical tasks. They bring order to chaos and aid in the discovery of patterns and structure.

· Development Team. The development team's main focus is on scope definition, although they work with the Domain Experts and Business Analysts in actual analysis.

Output of Initial Analysis

The primary output of Initial Analysis is the global feature set – the initial set of features, functions, and capabilities for the unit of work. This may be formally documented (if circumstances require), or more loosely defined. The important thing is to communicate the global feature set in ways that all participants can understand. Because individual participants have different needs and have preferred methods of communication, this typically requires multiple outputs (free-form text, use cases, high-level user stories, diagrams, models, verbal communication, presentations, index cards, emails, mock-ups, prototypes of working software, etc.).

Secondary outputs include a cost estimate, a cost-benefit analysis, and a high-level project schedule.

Feature Set Analysis

Feature Set Analysis (FSA) is performed continuously throughout the life cycle of a project. It's purpose is to build User Stories that feed into iteration planning and into individual iteration development. Most of the analysis effort on any project is performed as part of FSA, and it is the heart of the JITRA process. The specific tasks of feature set analysis include:

· Prioritization. The global feature set defined during Initial Analysis feeds the analysis of individual Feature Sets. At the start of Feature Set Analysis, all of the features, functions, and capabilities that have not yet been implemented (or which have only been partially implemented) are reviewed and prioritized by stakeholders. This process allows stakeholders – and not developers - to identify what parts of the system get the most focus, and always ensures that the most important part of the system will be analyzed and developed next.

· Selection. Once the remaining features, functions, and capabilities have been prioritized, the development team groups the highest priority items into a *Feature Set*. A Feature Set is nothing more than a grouping of the features, functions, and capabilities that the development team estimates can be analyzed and implemented in a small number of iterations (typically 2 – 4).

· Initial Story Analysis. The primary task of individual FSA is the initial analysis and definition of User Stories. After the Feature Set has been selected, Domain Experts and Business Analysts – supported by Requirements Stakeholders and the development team - perform an initial analysis of the items in the Feature Set. This analysis is used to build and define individual User Stories that feed into iteration planning and actual development.

· Iteration Planning. Initial User Stories are grouped into iterations and planning starts for these iterations. Close-in iterations are addressed first, followed by those farther out (at the point they are needed).

Scaling Feature Set Analysis

Scalability of Feature Set Analysis is not an issue. The size of a Feature Set is under the control of the development team and is always determined by how much of the remaining system can be analyzed and implemented in the next few iterations.

Concurrency of Feature Set Analysis

Feature Set Analysis is designed to be performed concurrently with actual development. Development team involvement in Feature Set Analysis – for tasks other than Feature Set selection – should be closely monitored so the development tasks of the current iteration are not impacted.

Selecting a Feature Set that is too big

In some cases, the team may discover the selected Feature Set is too big to fit into a few iterations. In these situations, simply focus on the highest priority items first, and return the unanalyzed items to the list of unimplemented features, functions, and capabilities. These items will then be addressed in follow-on Feature Set Analysis.

Outputs of Feature Set Analysis

The primary outputs of Feature Set Analysis are groups of initial User Stories. These User Stories should be detailed enough to allow follow-on iteration planning, yet they do not have to be detailed enough to implement (although they may be).

A secondary output of FSA is updated iteration plans. These identify what elements in a Feature Set are assigned to specific iterations and when delivery of the features can be expected.

Detailed Story Analysis

The purpose of Story Analysis is to finalize the details of each story *as close to implementation as possible*. This means that detailed analysis of stories should be deferred until the development team is ready to implement the story. This typically occurs at the start of iterations, but may occur at anytime within an iteration.

The specific tasks of story analysis are varied and beyond the scope of this article (for a comprehensive description of story analysis, refer to *User Stories Applied* by Mike Cohn [9]).

After-Action Analysis

At the completion of each iteration, an After-Action Analysis is performed. This allows "lessons learned" from the previous iterations to be included in the analysis process of subsequent iterations.

As part of After-Action Analysis, new requirements may be defined. These new requirements may take many forms:

- New items added to the list of features, functions and capabilities.
- Changes to the list of features, functions, and capabilities.
- New Stories added to the current Feature Set.
- New Stories that will be added to follow-on Feature Sets.

- Changes to baselined Stories that result in a Modified Story being added to the current Feature Set.

- Changes to baselined Stories that result in a Modified Story being added to the follow-on Feature Sets.

Regardless of the form they take, new requirements are fed back into the JITRA process at the appropriate time and at the level matching the understanding of the requirement.

4 HOW JITRA MEETS THE NEEDS OF DECISION MAKERS

As mentioned at the beginning of this article, all projects share the same basic goals. The following sections detail how JITRA addresses these objectives and how projects can benefit by implementing JITRA

Maximize Value

JITRA supports the maximization of value in several ways:

- Continuous Cost-Benefit Analysis. Every JITRA effort includes continuous cost-benefit analysis. This allows stakeholders to constantly asses and reassess value and communicate these assessments

- Prioritization of requirements. JITRA's continuous analysis process forces stakeholders to constantly identify the most important needs. This ensures the features having the greatest value are defined and implemented first.

- Quicker Time-to-Market. By implementing a concurrent engineering process that allows analysis and development to proceed in parallel, JITRA shrinks the time it takes to deliver software. Quicker time-to-market translates directly to added value.

Reduce Risk

JITRA uses an iterative approach to requirements analysis, and iterative approaches reduce risk [10]. Additionally, JITRA reduces risk by adhering to fundamental principles:

- Last Responsible Moment. By waiting until the final possible moment, JITRA produces requirements that are closely aligned with stakeholder needs. This alignment significantly reduces risk associated with three key risk factors:
  - Incomplete or inadequate requirements
  - Time wasted on meaningless requirements
  - Changing requirements
- Continuous Analysis. Constantly analyzing and refining requirements produce better requirements. Feedback from completed activities allows lessons-learned to be incorporated into future requirements and new requirements can be added, defined, and assed without elaborate change procedures.

- First-Things-First. By always focusing on the most important things, JITRA reduces the risk that the less important will impact the more important.

- Collaborative Communication. Constant, effective, and clear communication between all participants significantly reduces the risk that what is delivered won't be what is needed (or expected).

- Learn By Doing. JITRA expects analysis to be an active process. Where development is used to gain understanding as well as implement a solution. This reduces the risk of unknown issues cropping up late in a life cycle.

- Focus on Delivery. By focusing on delivery, JITRA reduces the risk of "analysis paralysis". Where time and money are spent analyzing the problem but no software is delivered.

- Equality of Value. By fixing cost and allowing features to fluctuate, JITRA reduces the risk of cost overruns.

Ensure Quality

JITRA ensures quality through the application of the following principles:

- Collaborative Communication. Errors due to miscommunication and differing expectations are all but eliminated using JITRA. Constant, clear, and effective communication throughout a projects life cycle reduces (if not eliminates) ambiguity – a major cause of quality issues for delivered software.

- Continuous Analysis. By constantly assessing and reassessing requirements - and by incorporating feedback into future analysis - JITRA limits the impact of change on a project. This results in higher quality software since errors can be fixed quicker and cheaper.

- Constant and Continuous Improvement. Feedback and refinement combined with an iterative analysis process facilitates continuous improvement. Over time, small continuous improvements always result in higher overall quality.

- Isolation of Complexity. By isolating complexity, JITRA also isolates errors. This limits the impact of errors and increases the quality of the overall system

5 SUMMARY

Just-in-Time Requirements analysis significantly reduces project risk and shortens development time. It ensures the most important parts of a system – as defined by the business stakeholders - are being worked on at any given point in time and only defines requirements when they are needed. It supports the evolution of requirements and provides mechanisms for easily incorporating changes into the analysis process. It shortens development time by continuously performing analysis concurrent with development rather than in sequence prior to development. In short, Just-in-Time Requirements Analysis matches the vision and promise of agile development and perfectly compliments agile development approaches.

REFERENCES

1. Cockburn, Alistair: *Agile Software Development*. Addison-Wesley. 2002. Page 31.

2. Lee, Michael: Just-in-Time Requirements Analysis: The Engine That Drives the Planning Game. XP2002

Conference.
http://www.xp2003.org/xp2002/atti/MichaelLee—Just-in-TimeRequirementsAnalysis.pdf

3. Cockburn, A: Page 148.
4. Cohn, Mike: User Stories Applied. Addison-Wesley. 2004. Page 1.
5. Highsmith, Jim: *Agile Methodologies: Problems, Principles, and Practices.* XP2001 Conference. http://www.xp2001.org/xp2001/conference/Details/AgileMethodologiesXP2001.pdf
6. Von Moltke, Helmut (The Elder). A common military saying originally attributed to Field Marshal von Moltke, Prussian Army, circa 1880.
7. Cohn, M: Pages 1-230
8. Cohn, M: Page 24.
9. Cohn, M.: Pages 1-230
10. Highsmith, Jim.

(Footnotes)

*

Of course, requirements specifications produce significant revenue for those consulting firms tasked with producing them. But that 's another article....

*

Although there is a raw sequence to these activities, they do not represent distinct phases in the analysis process. A sequence only applies to a specific time-slice and a given scope. Since JITRA expects both time-slices and scope to overlap, these activities actually define an ongoing, continuous process.

#

"Start" is defined as the point a decision is made to spend money on analysis and (possibly) development. Effort spent on analysis prior to the "start" is incorporated into the analysis process. In some situations it is possible to have all of Initial Analysis done before a unit of work has been started.

Managing Partner
Kuvera Enterprise Solutions
1750 30th Street, Suite 186
Boulder, CO 80301
(303) 638-7728
mike@kuvera.com

AGILE FOUNDATIONS

By Steve BerczukIndependent Consultantsteve@berczuk.comhttp://www.berczuk.com

To work effectively with Agile Methods you must have some basic skills. How well you master these skills can determine how successful you are in implementing your Agile Process. This section will help you to understand the traditional "foundation" skills that agile methods build on.

AGILE FOUNDATIONS: READ ALL ABOUT IT!

In the last issue we listed some general areas that every agile developer should know. In this issue I'll explore the general theme a bit further as a way of preparing for the workshop, and **Mike Cohn** will explore one particular area that is the foundation for all agile development: *User Stories*. In the next issue we'll explore another foundation area in depth.

The Five "W's"

Agility is about reexamining where you are and where you want to go – In short, asking questions is a key skill an agile developer should have. While most all questions can be useful, asking the right ones makes the process of growing, changing, and understanding your needs and your customer's needs, go more smoothly. Since stories play an important role in an agile process, I thought that it would be interesting to use a story as a framework for asking questions to help us to understand what agile development is built on. In this article we'll ask the five questions that every good *newspaper* story asks (though the need not be in 'inverted pyramid" style☺).

For those of you who have not read an introductory text on journalism, the questions that every newspaper story asks are, *who, what, when , where, why,* and also, *how*. In the context of "foundations for agile development" the questions take on a more detailed meaning:

% **Who** is involved with the project?

% **What** are we building?

% **When** are you delivering the project?

% **Where** are the participants on the project located?

% **Why** are you building this project?

% **How** are you are you making technology and process choices for your project?

Keeping these ideas in mind is especially important when you are trying to move from your current process towards an agile process; it will help you to focus on whether you are building a good foundation while you are moving towards your goal. And keeping these questions in mind will help you to keep decision makers informed of what you are doing in a manner they are familiar with.

Understanding People (Who)

Team members, customers, the people in the surrounding organization are all important to the success of an agile project. Agile processes emphasize the role of people in a successful process more than most other processes, but interpersonal skills can not be understated.

Agile projects depend heavily on the team members. While all projects are composed of people, the agile teams rely heavily on communication (*Individuals and Interactions*) to be successful. A common stereotype is that people who choose computing as a profession do it to (in part) avoid extensive interaction with others. This is not a universal truth, but much of the work of generating code is spent in solitary work, or team work with other "software people." Understanding how people (and this includes, yourself) is extremely helpful in all aspects of an agile process, including understanding and developing requirements, and communicating among team members.

Other contributors to *The Agile Times* have more to say about the people dimension.

Understanding the Target (What)

Understanding what to deliver to your customer is essential to an agile process since you are delivering software in small frequent iterations, and each iteration needs to demonstrate value.

Mike Cohn's article "Questioning your Users" discusses this in more detail so I won't say much more here.

Timing (When)

Agile processes deliver a working, if incomplete, product to your customers at the end of every iteration. To do this well, an understanding of interaction design will help you to build systems that customers can look at each new iteration without extensive training. Some resources for this are *Designing Web Usability* [1], and the: *The Elements of Friendly Software Design* [2] by Paul Heckel.

Since agile processes discuss incremental development, some believe that architecture is counter to the ideals of agile processes. In fact, how you structure your architecture and release process are essential in maintaining that. Keeping a good project rhythm is essential to sustaining an agile process.

For more about Rhythm, see the book *Software Architecture : Organizational Principles and Patterns*[3].

Location, Location, Location (Where)

Agile methods rely heavily on collocation of the team and the customer. Sometimes in spite of our best intentions the customer may not be near the team, and the entire team may not be in the same location. While this may reduce the effectiveness of the process, you can lessen the problems by:

% Designs which allow for some work to be done independently,

% Appropriate configuration management to allow the teams to exchange code.

I agree with the idea that distributed teams make *any* development effort difficult, and since agile teams move rapidly communication is even more important. But especially in situations where you are trying to introduce agility into a situation, you have to work with what you have.

Why?

An agile process gives us a chance to demonstrate value more quickly, and in the end we hope that we are delivering software to add value to a project or company. Unfortunately there are projects for which the team can't answer the question of 'why' they are working on them beyond the immediate "someone is paying for it." You can build systems like this but it's hard to build good systems from this mindset. Recently Mary Poppendick, author of Lean Software Development, met with a group of Boston Area agile developers and discussed the value of domain understanding in software development. Understanding is key: Developers who understand what they are building will build better systems because they can more effectively bridge the gap between the customers wishes and what can be built with the technology to satisfy the customer's needs.

How?

The answer to the "how" question is where most of the unsung foundation skills show up. Who, What, When, Where, and Why are, for the most part, addressed by the principles and practices of the agile processes, but there is a long way from knowing a set of agile principles and practices are using them effectively. Frequent integration and frequent delivery is necessary for many agile processes to work, but to actually do this integration effectively you need to know something about build management, release engineering, and creating and managing development workspaces. What is especially important is knowing what aspects of these disciplines to take and which to leave. A big problem is that these words get overloaded to mean more than they should. This is a subject for another day.

SUMMARY & LOOKING AHEAD

I hope that the first 2 articles of the Agile Foundations section have gotten you thinking about what background a team needs to successfully execute an agile project. At the Agile Development conference in June, I'll be facilitating a workshop where we'll be developing this concept a bit further. If you have any suggestions for topics for this column, or want to contribute to the next issue, send me a note: steve@berczuk.com. In the next issue one we'll discuss some basic software build and workspace management skills that can help you be more agile.

References

1. Nielson, J., *Designing Web Usability*. 2000, Indianapolis, IN: New Riders.
2. Heckel, P., *The Elements of Friendly Software Design*. New ed. 1991, San Francisco: SYBEX. xxx, 319 p.
3. Dikel, D.M., D. Kane, and J.R. Wilson, *Software Architecture : Organizational Principles and Patterns*. 2001, Upper Saddle River, NJ: Prentice Hall.

About the Author

Steve is a software developer and consultant based in Arlington, MA. He is the author (with Brad Appleton) of the book Software Configuration Management Patterns: Effective Teamwork, Practical Integration. Steve has published articles in the areas of patterns, software teams, and architecture. Steve has built software systems in domains ranging from satellite telemetry processing applications to internet applications to CRM and supply chains systems. He has an M.S. in Operations Research from Stanford University and an S.B. in Electrical Engineering from MIT. You can contact him at steve@berczuk.com. His web site is www.berczuk.com.

QUESTIONING YOUR USERS

By Mike Cohn

User stories are a foundational element of XP. They have also been successfully used with other agile processes, most notably Scrum. User stories shift focus from written requirements documents to conversations between customers and developers. Because of this, it is important that we get as much as we can from these conversations, which means that the questions we ask—and how we ask them—are vitally important. It is never sufficient to ask the user “So, what do you need?” Most users are not very adept at understanding, and especially at expressing, their true needs. I learned this once from a user who walked into my office and acknowledged, “You built exactly what I asked for but it’s not what I want.”

Another time, I worked with a team that was developing software for delivering surveys. Each survey would be delivered over the phone, via email, and via interactive voice response. Different types of users would use different survey types. The surveys were very complicated: specific answers to one set of questions would determine which question would be asked next. For example, based on answers to body height and body weight questions, the system would calculate the respondent’s Body Mass Index (BMI) and then invoke a page based on whether the respondent was overweight.

The users needed a way to create the surveys. When the users first told the developers about their need, they presented us with examples of a complicated mini-language. They proposed typing surveys in this new language. This entirely text-based approach seemed needlessly complicated to us. We asked the users a few questions about how they’d use the new system and it became clear that a drag-and-drop survey-building GUI would be better for them than their proposed new language.

Having a problem does not qualify you to solve it. In this case, the users who had the problem were proposing a solution that was needlessly difficult. It took the developers carefully questioning the users to pull out their true needs.

The best technique for getting to the essence of a user’s needs is through the questions you ask. I worked with a project team that was torn between putting their application in a browser or writing it as a more traditional platform-specific program. They struggled between the ease-of-deployment and lower training costs provided by the browser-based version and the more powerful platform-specific client. The intended users would certainly like the advantages of the browser, but they also valued the richer user experience provided by the platform-specific client.

It was suggested that the target users for the product be asked their preference. Since the product would be a new generation rewrite of a legacy product, the marketing group agreed to contact a representative sample of current users. Each user in the survey was asked “Would you like our new application in a browser?”

This question was like going to your favorite restaurant and having the waiter ask if you’d like to have your meal for free. Of course, you would! And of course the surveyed users responded that they would love to have the new version of the software in a browser.

The mistake the marketing group made was that they asked a *closed-ended question* and failed to provide sufficient detail for it to be answered. The question assumed that anyone being interviewed would know the tradeoffs between the browser and the unstated alternatives. A better version of the question would have been:

> Would you like our new application in a browser rather than as a native Windows application even if it means reduced performance, a poorer overall user experience, and less interactivity?

This question still has a problem because it is closed-ended. The respondent is given no room for anything other than a simple yes or no. It is far better to ask open-ended questions that let respondents express more in-depth opinions. For example, "What would you be willing to give up in order to have our next generation product run within a browser?" A user answering that question can go in a variety of directions. Where she goes—and does not go—with her answer will provide you with a more meaningful answer to the question.

It is equally important to ask context-free questions, which are ones that do not include an implied answer or preference. For example, you would not ask, "You wouldn't be willing to trade performance and a rich user experience just for having the software in a browser, would you?" It's pretty clear how most people are going to answer that question.

Similarly, instead of asking "How fast do searches need to be?" ask "What kind of performance is required?" or "Is performance more important in some parts of the application?" The first question is not context-free because it implies there is a performance requirement to searching. There may not have been but, having been asked, a user is unlikely to say so; she's more likely to take a guess.

At some point you will need to move from context-free questions to very specific questions. However, by starting with context-free questions you leave open the possibility for a wider range of answers from users. That will lead you to user stories that may have remained undiscovered if you jumped right into very specific questions.

Mike Cohn (mike.cohn@computer.org) has twenty years of experience developing software and is the Vice President of Engineering at Fast401k. He is a founding member and on the Board of Directors of the Agile Alliance. Some ideas in this article are taken from his most recent book, *User Stories Applied: For Agile Software Development.* He has also written books on Java and C++ programming and is a frequent conference speaker.

ACHIEVING "FLOW" THROUGH AGILE DEVELOPMENT

Scott Bogartz

In 1991, Mihaly Csikszentmihalyi coined the term "flow" to describe a common mental state experienced by people when they feel a deep sense of enjoyment.[1] Since then this state has been linked to individuals reaching peak performance in many different areas including business, creative arts and sports. The concept resonated with many because it described feelings they had actually encountered during their own best performances. And it resonated because it had been described in various terms for over a century in psychology, literature and religion.[2] The goal of achieving flow became a hot topic among many executives. This was especially true after Super Bowl winning coach Jimmy Johnson was featured discussing flow in his approach to coaching and life.

This series of articles is about convincing senior management that Agile development is an extremely powerful tool for achieving great results. While one goal of the series is to present empirical evidence of Agile successes, another is to help its proponents better articulate how Agile "just makes sense". By comparing the principles of Agile to patterns observed in nature, production theory, philosophy and psychology, we can make a compelling case for Agile. And we can do so in terms with which senior executives can connect. Flow offers an exceptional opportunity for such a connection. It is broadly recognized and studied in the business management community with many books and symposiums offering advice on how to achieve it in business, personal life, community affiliations and sports. In examining the components of flow one can see a good deal of parallelism to the practices and motivations of Agile development. In the following paragraphs I will present the key components of flow and describe how they relate to Agile development. I categorize the components as either enablers or desired results as they relate to Agility. Enablers allow an organization to reach its optimal performance level, and desired results are the traits that drive superior execution.

The first enabling component of flow is that "goals are clear".[3] Here the goal refers to the immediate task at hand and not the ultimate result of a sequence of tasks. For example, an athlete must focus on executing the current play to the best of her abilities. Too much worry about winning or losing the overall contest will distract her from top performance. We see this enabler in Agile development in that delivering short iterations and responding to changing demand is valued over adhering to an overall plan or building to a pre-conceived set of functionality. The goal of each iteration is clear to the team. The team is able to respond and adapt without worrying too much about the overall plan. In contrast, when a team is focused on a pre-determined plan, activities such as explaining (or worse, hiding) schedule variances and battling to prevent changing requirements distract from the desired result of delivering business value.

Another enabling component of flow is that "feedback is immediate". An individual is best able to stay absorbed in an activity that provides a continual stream of information about the quality of her performance.[4] The concept of immediate feedback permeates Agile development. Practices such as frequent delivery to customers, daily meetings, continuous integration and automated testing provide feedback at several levels. The feedback loop includes all effective people from individual developers through management and stakeholders. The feedback allows a team to make the minute adjustments necessary to stay on course (instead of making drastic changes to get back on course). It also provides the satisfaction of a "job well done" as a continual benefit. Finally it promotes confidence among the team and stakeholders so that everyone stays absorbed and committed.

The final enabler that I will discuss is that there exists a "balance between opportunity and capacity".

Tasks that are perceived as infeasible generate anxiety and guilt which interfere with peak performance. Flow occurs when both challenges and skills are high and equal to each other.[5] Under Agile practices, software development provides a wonderful environment for fostering flow. When a well skilled team engages in a project that is complex yet achievable, there is high potential for flow. Being forced to accept unrealistic timelines or constraints (lack of technology, training, etc) will make successful completion seem out of reach and make the team feel its efforts are ultimately useless. Tasks must have meaning in order to promote flow. Trying to provide the (false) notion that the complexity can be removed from a software development effort by following a strict, standardized process is actually counterproductive. It belittles the skills of the team and replaces the feeling of invigoration that comes with accomplishment to one of tedium.

When the environment is right and flow occurs individuals (and organizations) display several positive traits that power peak performance. The most important trait is that "concentration deepens".[6] The actor becomes one with the activity. Execution seems effortless, though the individual is navigating a complex array of challenges. As developers and teams, I know many of us have experienced this feeling. When the distractions and interruptions are removed and we are able to focus on creating simple, elegant solutions to complex problems, our concentration is total. We do our best, most satisfying work in those intense periods. I can count numerous occasions that I have broken through a days-old mental log jam during such a state of increased concentration and felt as if the answer "just came to me". What executive would not want his software developers to spend more time in such deep concentration?

Other flow components that fit into the desired result category include that "control is no problem" and "the sense of time is altered".[7] In a flow state, the mind is cleared of the worry, doubt and distraction that often interfere with performance. The sense of time changes to accommodate the needs of the situation. Time might slow down to allow one to make a critical decision or speed up to allow one to stay absorbed in a task for longer than she realized. These traits are extensions of the deeper concentration achieved during flow. They are very conducive to problem solving, creativity and mentally demanding tasks like software development.

In this article, we have seen that several of the key practices of Agile development are enablers of flow at the individual and organizational level. We have discussed the behavioral traits associated with being in a flow state and how those traits are beneficial to software development. Knowing that business leaders are interested in achieving flow in many areas, it makes sense that they would support a software development approach that is consistent with flow. In addition to the benefits already listed, it is important to note that activities that generate flow are considered worth doing simply for their own sake.[8] Hopefully the line of reasoning presented in this article will help a few of us connect Agile development to better focused, more creative and more motivated teams. This should be a powerful selling point to present to senior management.

Good Luck.

[1] Mihaly Csikszentmihalyi, *Good Business: Leadership, Flow, And The Making of Meaning* (New York: Penguin, 2003) 39.

[2]

Csikszentmihalyi 59-61.

[3]

Csikszentmihalyi 42.

[4]

Csikszentmihalyi, 43.

[5] Csikszentmihalyi, 44-46.

[6] Csikszentmihalyi, 48.

[7] Csikszentmihalyi, 50-57.

[8] Csikszentmihalyi, 56-57.

Waltzing with Bears - Tom DeMarco and Timoty Lister

Alex Pukinskis

WALTZING WITH BEARS: MANAGING RISK ON SOFTWARE PROJECTS BY TOM DEMARCO AND TIMOTHY LISTER

You might be familiar with the famously late baggage-handling software project that delayed the opening of Denver International Airport, leading to over a million dollars a day in excess financing costs during the delay. The project is often considered the classic example of how bad the software industry is at delivering projects on time. In their new book, *Waltzing With Bears*, Tom DeMarco and Timothy Lister argue that because the software contractor consistently identified the schedule as unrealistic from the beginning, the delays were not a result of an inadequate software development process so much as a failure of risk management on the airport project as a whole.

The 2004 Jolt Award winner *Waltzing With Bears* is a sound introduction to risk management on software projects. DeMarco and Lister present the case for risk management, declaring that risky projects are the only ones worth doing. Risk management can bound uncertainty, focus attention where it is needed, and prevent projects from getting blindsided.

The authors readily acknowledge that there are circumstances where risk management is not appropriate. In a "manage for success" culture, identifying risk and uncertainty can quickly get you fired. I found a lot of overlap here with the honesty and truth-telling necessary for successful adoption of agile processes.

All too often, our software schedules reflect only the "best-case" dates, which in reality we only have a small chance of meeting. DeMarco and Lister provide risk diagramming techniques that help to quantify uncertainty in delivery dates, allowing us to identify the most likely date in addition to the most optimistic ones.

The book gives us some methods to use in the risk discovery process, so we can maintain a list of project risks. Many of us have done this on our projects, only to experience pressure to eliminate the risks over time, shrinking the list like it's a set of incomplete requirements. In reality, only certain kinds of risks expire, so each risk needs to be managed uniquely; we need to choose between avoiding, containing, mitigating, and evading that risk.

Of particular interest to me was their discussion of the five core risks inherent to software projects: inherent schedule flaw, scope creep, employee turnover, specification breakdown, and poor productivity. Certainly many of these risks can be managed through the use of agile techniques, and indeed, the authors are very enthusiastic about benefits of using incrementalism to mitigate certain types of software project risks.

DeMarco and Lister suggest that value quantification should play a key part in risk management, because we need to know what kind of benefit we stand to gain in return for a given investment. "We gotta have it to survive" isn't good enough anymore, they say, and they rightly point out that many "death-march" projects are treated as essential precisely because their benefit hasn't been quantified (and doesn't exist!). What's missing from the book is some solid guidance in calculating benefit on non-direct-cost-saving systems.

While the book is great overall, I have a couple of minor criticisms. Midway through the book, we're provided with a nine-step "risk management prescription": basically, steps you can take to do risk management on your project. Then at the end of the book, we see the list again, revised with 18 steps now that reflect the intervening chapters. We're not really told how to choose between the lists; does the second one supplant the first? Also, some readers will probably feel too much attention is given to basic statistics; the rest of us will appreciate being reminded of material we might not have seen since college.

In the software industry, we often see our development processes take the blame for failed or late projects. Reading *Waltzing with Bears* reminds us that sometimes a well-optimized agile team is not sufficient to guarantee success; without risk management, our projects can easily become irrelevant.

-Alex Pukinskis

Waltzing with Bears: Managing Risk on Software Projects by Tom DeMarco and Timothy Lister Dorset House $27.95

User Stories Applied - Mike Cohn
J.B. Rainsberger

REVIEW OF USER STORIES APPLIED BY MIKE COHN
BY J. B. RAINSBERGER

Mike Cohn's User Stories Applied provides a thorough examination of user stories, including what, how and why. As an experienced XP/Agile software professional I initially indulged in the conceited notion that I would stand to learn little from such a book; after all, I had made user stories a normal part of all my projects since 2000. How much could I possibly not know? Well, there was plenty left to learn.

Mike's breadth of experience shines through in every part of this book. As he explains user stories he provides numerous comparisons to other commonly-used approaches to requirements gathering and documentation. Like a good programmer, Mike provides more than just the happy paths – he admits that user stories are no silver bullet and concedes those cases where more formal requirements gathering and documentation are to be preferred. One of the key benefits of this book is that it treats a wide variety of contexts in which user stories play a role: requirements gathering, documentation, communication, short-term planning, long-term planning, risk management and a few I probably just can't remember. Paradoxically, this thin volume is a compendium of user stories knowledge. How does he do it? He gets to the point.

The writing style is lively and paragraphs are densely packed with useful information. Each chapter ends with not only an executive summary, but a small number of questions to help the reader apply what they have learned. User Stories Applied seems to be a "poster book" for the "no fluff, just stuff" movement. Mike weaves enough real-life experience to provide concrete examples of user stories at work to be instructive without detracting from the book's overall pace. Beyond that, the book surprised me – pleasantly, of course – with some content I hadn't expected: a catalog of user story "smells," a look at user stories and Scrum and "What user stories are not." These portions of the book, combined with the well-developed example in Part IV, justify the word "applied" in the title.

I came away from this book with a deeper understanding of what constitutes a good user story. This alone is easily worth the time spent reading. The catalog of smells is something I expect to refer to just as frequently as I refer to Martin Fowler's classic Refactoring. If you are having trouble taming requirements, then this book will certainly help you get back on the right track.

User Stories Applied for Agile Software Development by Mike Cohn Addison-Wesley $34.99

Volume V, Issue 1

Values, Power, and Agility

David Hoover, ThoughtWorks

Dave Hoover, dhoover@thoughtworks.com

When a software development team adopts agile principles, [MANIFESTO] it is not uncommon for the team to experience conflicts between its newly adopted principles and the values of its parent organization. When teams and organizations have opposing values, power struggles will inevitably erupt. The purpose of this article is to generate a greater awareness of power distributions and value conflicts among agile teams and their parent organizations. It is my belief that an increased awareness of these conflicts will help these groups collaborate more successfully.

As teams transition toward agility, power is necessarily redistributed. The general theme of this power redistribution is that of empowerment of the people on the front line:

Ø Developers provide estimates for their work and have authority to evolve any aspect of the system.
Ø Architects are in the trenches, actively tending to the system design.
Ø Testers provide automated tests that drive the development process from the outset of the project.
Ø Customers possess the ability to steer the project, allowing them to adapt their requirements as their business needs change.

Agile teams experience the powerful phenomenon of collective mind. [WEICK] This phenomenon is the result of a richer interrelating among teammates and the inclusion of experienced and novice people together on the same team.

To maintain agility, it is critical that an agile team keeps a watchful eye on how its values are interacting with the values of its parent organization. This article is peppered with questions to help raise the reader's awareness of the power struggles within which these value conflicts hide.

- *What are the explicit values of your team? What are the implicit values of your team?*
- *What are the explicit values of your parent organization? What are the implicit values of your parent organization?*
- *Does your team possess the authority to direct itself? What stands in the way of your team becoming self-directed?*

Merriam-Webster's web site defines "power" as the following...

> power (*noun*): 1) the ability to act or produce an effect, 2) possession of control, authority or influence over others.

For the remainder of this section, I will refer to the above definitions as *first-order power* (the ability to act or produce an effect) and *second-order power* (possession of control, authority or influence over others). In the context of software development, my understanding of first-order power is the power to directly act or affect the system under development, such as writing code. I view second-order power in this context as the power to influence the direction of the team.

- *What sorts of power do you possess in your everyday work?*
- *How have the values of your team influenced the types of power you possess?*

Developers

On a traditional team, a developer has first-order power. She has little authority over anyone or anything other than the functionality she is currently developing. Even the small amount of authority she has might be subjected to the will of the architect, depending on the precision of the design specifications.

Developers, as a whole, gain more power than any other role on an agile team. While they retain first-order power over the code they are currently developing, they are explicitly given second-order power to evolve the best possible design in continual communion with their fellow developers. It is in this communion that individual developers seemingly lose some of their first-order power. Rather than retaining exclusive rights to specific components, their authority is decentralized into the collective of developers. [CCO-WIKI]

This decentralization of authority into the development team enables a phenomenon known as collective mind. [WEICK] While the first-order power of the individual developer appears to diminish, the power of the collective is greater than the sum of its parts. In their study of flight deck teams on aircraft carriers, Karl Weick and Karlene Roberts found that

> "No matter how visionary or smart or forward-looking or aggressive one brain may be, it is no match for conditions of interactive complexity. Cooperation is imperative for the development of [collective] mind." [WEICK:378]

This finding is congruent with the experiences of successful self-organizing teams. While variations in ability exist among the teammates, there is a consensus that no one member could have designed as good a system on her own. Thus, via collective mind, an agile developers' first-order power has increased in breadth, if not depth.

Agile developers are granted second-order power to estimate their own development tasks. This is a critical point of change for many teams who traditionally had team leads, architects, and/or management handing down fixed-length, fixed-scope schedules to developers. Second-order power, in this circumstance, does not necessarily mean that developers have power and authority over project managers and architects, but that the power between these roles has been balanced.

- *With the adoption of agile principles, how has power shifted within your development team? Where have power struggles emerged?*
- *Would you say that your development team posesses a collective mind?*

Architects

> "Exceptional designers exercise leadership through their superior knowledge rather than bestowed authority." [POPPENDIECK:112]

On an agile team, an architect's first-order power has increased. Where a traditional architect might deliver blueprints to a team of developers, an agile architect acts as a guide, [FOWLER] rubbing elbows with developers and getting his hands dirty in the implementation of the system. He has immediate and ongoing access to the state of the system's design, whereas in a traditional environment, he might be hierarchically removed from the implementation.

An agile architect's second-order power remains unchanged, though it is manifested differently. While a traditional architect's second-order power is often imposed through hierarchical channels, an agile architect's second-order power is exercised through face-to-face collaboration with his teammates. An agile architect's authority is less likely to be experienced as coercive or authoritarian than a traditional software architect. His authority is now established by his abilities to communicate and contribute rather than through simply holding a position of power within the organization.

The need for agile architects to descend into the trenches can become a stumbling block for organizations in which architects are inexperienced in system implementation. When architects are ignorant to how to implement their own ideas, their value on agile teams is diminished. With a decreased ability to exercise second-order power via hierarchy and the inability to exercise first-order power at all, these architects will likely resist a move toward agile principles.

Mary and Tom Poppendieck use the term *master developer* to label the role that people traditionally think of as architect:

> "Master developers are part of the team, enmeshed in the details of the work. They provide the leadership necessary for the team to make good decisions, make rapid progress, and develop high-quality software." [POPPENDIECK:113]

Clearly, the common thread for the agile architect is his inclusion as a member of the development team. His inclusion is a critical enabler of collective mind and the steady improvement of the team. Weick and Roberts found that "[a]s seasoned people become more peripheral to socialization, there should be a higher incidence of serious accidents." [WEICK:368] There are few better ways for novice developers to improve than to work closely with architects. [COCKBURN-WIKI] Interestingly enough, architects will find that they too will benefit from these interactions: "Comprehension can be increased if more levels of experience are connected, as when newcomers who take nothing for granted interrelate more often with old-timers who think they have seen it all." [WEICK:366]

- *How do architects in your development organization interrelate with developers? Would these architects fit the description of guide, mentor, or master developer?*
- *Do your organization's architects lack first-order power? How does this lack of first-order power affect the interactions between developers and architects?*
- *What does "architect" mean in your organization?*

Testers

The relationship between developers and testers has traditionally been antagonistic. Developers work to produce functionality while testers work diligently to uncover flaws in this functionality. It is not uncommon for testers to discover defects that developers feel are simply misunderstood requirements. This can then develop into a second-order power struggle over who holds the authority to interpret requirements. This power struggle will almost inevitably bubble up to the customer, diminishing his confidence in the development process.

Agile principles call for software to be released frequently, preferably, every couple weeks. [MANIFESTO] To achieve this feat, an agile tester must test functionality in frequent bite-sized chunks rather than in huge batches. The ability to embrace changing requirements and complete iterative test cycles requires the agile tester, like the agile developer, to collaborate continually with the customer. Agile testers interpret the customer's requirements and develop tests to drive the development effort.

Robert "Uncle Bob" Martin wrote of the power shift between developers and testers:

“[Agility] completely changes the power structure between development and QA... QA finds itself in a specification role as it writes the acceptance tests that define the features. Development can no longer dump a pile of crap on QA. Instead QA dumps requirements on development.” [UNCLEBOB]

Thus, a tester, traditionally a person with only a small amount of indirect first-order power, now finds himself to be the cornerstone of the agile software development effort. While his first-order power remains indirect, this power is far more focused and prescriptive, rather than delayed and reactive. Because the agile tester is the primary supporter of the customer’s requirements, his second-order power is equal to that of the customer. The agile community has made significant progress toward integrating testers into the agile team and I believe testers will find themselves with far greater power because of these efforts.

- *How are testers perceived in your development organization? How are testers perceived on your team? Are testers considered part of the development team?*
- *At what point and to what degree are testers involved in your development process? How do these factors impact the power of the testers? How do these factors impact the developers’ relationships with the testers?*
- *Is there a distinction between developer and tester on your team?*

Project Managers

A project manager rarely finds herself in a position of first-order power. Her role tends to prevent her from contributing directly to the project. While an agile project manager is no more likely to write code than any other project manager, she does tend to focus herself more on removing obstacles that stifle the team’s ability to make progress, [CCPACE] a sort of indirect first-order power.

The agile project manager’s focus on obstacle removal does not lessen her second-order power, though it does tend to deemphasize it. In many organizations, she retains her traditional second-order power, but exercises it sparingly. The agile project manager tends to defer her second-order power to the development team, recognizing that, “the more the team relies on outsiders to make its decisions, the less control it has over its commitments.” [SCRUM:45] Thus, an agile project manager recognizes that she is an outsider, and takes on the roles of development facilitator, organizational liaison, and visionary leader. [CCPACE]

- *As the liaison between your team and the organization, what value conflicts does your teams’ project manager experience on a regular basis?*
- *Do the organization’s expectations of your team’s project manager conflict with the agile principles?*

Customers

The customer gains significant second-order power on an agile team. Rather than being kept at arm’s length in a traditionally antagonistic relationship with the development team, the agile team depends on frequent face-to-face collaboration with its customer. While the customer does not have absolute authority to dictate what should be delivered when, he regularly exercises his power to prioritize what should be delivered next.

The agile principles have introduced a fundamental shift in perspective on changing requirements, a historically favorite complaint of the software development industry. Agile teams embrace late changing requirements, viewing them as opportunities to increase the value of the system. [MANIFESTO] Thus, agility increases the customer's second-order power not only in depth, but also in breadth.

- *Would your team's customers agree that their power has increased since your transition to agility? What was it that facilitated this increased power for your team's customer?*

THE POWER OF THE FAMILIAR

"The power of the familiar is very strong, often stronger than the wish to change. Strong interventions, lots of patience, and continual awareness help us challenge the power of the familiar." [SATIR:212]

One of the chief frustrations with adopting agile methodologies is that the values set forth by an agile approach often stand in direct opposition to the implicit values of the larger organization. Even if a team is successful at integrating the principles of agility, the values and practices of the larger organization will act as a consistent and powerful force, pulling the team back toward the values of the organizational system.

A development team in an organization is not unlike a child in a family. While a child may choose to adopt a lifestyle that contradicts the traditions of the family, it is unlikely that the family is going to suddenly embrace that lifestyle as its own. It is more likely that the child will gradually revert to her family's traditions. This likelihood has much to do with Virginia Satir's notion of the power of the familiar:

"Familiarity exerts a powerful pull. What we have observed and experienced day after day exerts a powerful influence. Most people will choose the familiar, even though uncomfortable, over the unfamiliar, because of that power." [SATIR:144-145]

Beyond the power of the familiar lies the child's own power within the system. As a child grows physically and matures psychologically, her power will necessarily increase. Eventually, the power of the adult child will rival that of her parent. Yet, as the child grows in power, the power of the familiar grows with her, making it increasingly difficult for her to make fundamental changes in her lifestyle.

Similarly, a software development team that chooses to adopt a process that defies the values of its organizational system will not only face resistance from the power of the familiar, but also from the power of the organization itself. Like an adult child in a family, a team that holds a position of power within the organization will likely see its newly adopted agile principles spread into the organization. Conversely, if the organization is very large or the team immature, the team will likely struggle to maintain its newly adopted principles and gradually revert to the principles of the organization.

"A single new member, or even a single new group within so large a group, really has no chance of converting the social system, even if he is firmly convinced of the correctness of his way of doing things." [WEINBERG:63]

An exception to this idea occurs when a team has an executive sponsor, a champion that shields them from organizational pressure. Imagine what a difference it would make to have a well-respected uncle stand up for a child in the midst of a family conflict. This high-level support can be a critical enabler of the adoption and maintenance of agile values within a traditional organization.

In most cases, the success of a software development team's transition toward agility will be dependent on transitioning the parent organization concurrently. This may mean that the team will have to persistently engage its parent organization in order to spark an organizational transition. Alternatively, whatever force initially triggered the transition in the team may need to work concurrently in the organization. Either way, it is vital that both the team and the organization stay aware of how power is shifting as hierarchies collapse and agile values emerge.

- *During your team's transition toward agility, when was it most difficult to resist the power of the familiar? How was the team able to establish new principles and practices in the face of the power of the familiar?*
- *How much power does your team hold within the organization?*
- *Using the metaphor of the child and the family, how would you describe your team?*

THE POWER OF AWARENESS

> "In habitual action, each performance is a replica of its predecessor, whereas in heedful performance, each action is modified by its predecessor." [WEICK:362]

Families that struggle with change and conflict sometimes employ family therapists to help the family improve itself. Similarly, organizations struggling with conflicting values or the inability to adapt can benefit from hiring outsiders to facilitate change.

Agile teams meet at regular intervals to reflect on how to become more effective. [MANIFESTO] A retrospective [KERTH] is a common agile practice that is often facilitated by someone outside the system, though internal consultants or coaches can facilitate them as well. While an entire organization does not usually attend a retrospective, the process includes anyone that the team interacts with on a regular basis.

Retrospectives, when done regularly and successfully, can significantly increase an agile team's awareness of how its values are integrating into the organizational system. Furthermore, by reflecting on its own experiences, both the power of the familiar and the distributions of power become more visible to the team. This increased awareness provides an agile team with the ability to adapt and evolve itself, along with engaging its parent organization in intelligent dialogues about problematic value conflicts and power struggles.

- *How well do your team's values integrate with the values of the agile manifesto?*
- *How well do the values of your parent organization integrate with the values of your team?*
- *What value conflicts lie beneath the surface of the power struggles your team experiences?*

CONCLUSION

The power of agility is seen in the empowerment of the front line workers by releasing authority from hierarchical structures.

> "The people on the front line combine the knowledge of minute details with the power of many minds. When equipped with necessary expertise and guided by a leader, they will make better technical decisions and better process decisions than anyone can make for them." [POPPENDIECK::xxvi-xxvii]

When value conflicts exist, power struggles will surface and the power of the agile principles will be diminished. To maintain agility, it is critical that an agile team keeps a watchful eye on how its values are interacting with the values of its parent organization.

ACKNOWLEDGEMENTS

Patrick Morrison and Laurent Bossavit provided excellent guidance and feedback on this article. Rebecca Parsons and Boris Gloger brought up some important issues that I had missed. Thanks for your

time everyone.

REFERENCES

[CCO-WIKI] http://c2.com/cgi/wiki?CollectiveCodeOwnership

[CCPACE] Sanjiv Augustine and Susan Woodcock, *Agile Project Management*, http://ccpace.com/Resources/documents/AgileProjectManagement.pdf

[COCKBURN-WIKI] Alistair Cockburn, http://c2.com/cgi/wiki?ExpertInEarshot

[FOWLER] Martin Fowler, *Who Needs an Architect?* http://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf

[KERTH] Norm Kerth, Project Retrospectives

[MANIFESTO] *Principles behind the Agile Manifesto*, http://agilemanifesto.org/principles.html

[POPPENDIECK] Mary and Tom Poppendieck, Lean Software Development

[SATIR] Virginia Satir, The New Peoplemaking

[SCRUM] Ken Schwaber and Mike Beedle, Agile Software Development with Scrum

[UNCLEBOB] Robert Martin, March 12, 2004, http://groups.yahoo.com/group/chicago-agile-dev/message/1546

[WEICK] Karl Weick and Karlene Roberts, "Collective Mind in Organizations: Heedful Interrelating on Flight Decks," Administrative Science Quarterly, 38 (1993): 357-381

[WEINBERG] Gerald M. Weinberg, The Psychology of Computer Programming

Marcel Proust Questionaire

Rachel Davies

Rachel is a Certified Scrum Master who lives in the town of Rugby in the UK. She is interested in finding ways to help teams work more effectively to achieve their goals and has found agile development practices to be highly effective. She learned a lot about XP at Connextra but now Rachel is an independent consultant specialising in agile coaching. Rachel has presented reports and workshops at international conferences on software process most recently OT2004, XP2003 and ADC2003. She helps run eXtreme Tuesday Club in London and put together programs for international agile conferences - such as XPDays, ADC2004 and XP2004. Rachel is one of the Agile Alliance directors.

Her Answers:

1. What do you regard as the lowest depth of misery?
Being alone
2. Where would you like to live?
In Wales by the sea
3. What is your idea of earthly happiness?
Being at home with my children
4. To what faults do you feel most indulgent?
My own
5. Who are your favorite heroes of fiction?
Peter Pan
6. Who are your favorite characters in history?
Socrates
7. Who are your favorite heroines in real life?
Boudica
8. Who are your favorite heroines of fiction?Jane Eyre

Marcel Proust Questionairre
Rachel Davies

9. Your favorite painter?
Claude Monet
10. Your favorite musician?
Alan Stivell
11. The quality you most admire in a man?
Not following a stereo-type
12. The quality you most admire in a woman?
Leadership
13. Your favorite virtue?
Honesty
14. Your favorite occupation?
Gardening
15. Who would you have liked to be?
Architect
16. Your most marked characteristic?
Perseverance
17. What do you most value in your friends?
A cheery smile
18. What is your principle defect?
Criticising others
19. What historical figures do you most despise?
Adolf Hitler
20. What natural gift would you most like to possess?
To play music by ear
21. How would you like to die?
Unexpectedly
22. What is your present state of mind?
Busy
23. What is your motto?
Ad astra per aspera

Marcel Proust Questionairre
Alistair Cockburn

A short bio of Alistair:(I got this from his website (http://members.aol.com/acockburn/wwwbio.html):

Alistair Cockburn is consulting fellow at **Humans and Technology**. He is responsible for helping clients succeed with object-oriented projects, including: corporate strategy, project setup, staff mentoring, process development, technical design and design quality. Mr. Cockburn has over 20 years of experience, leading projects in hardware, software, research, and application systems. He is an internationally recognized expert on object-oriented software development, and appreciated as a collaboration facilitator.

And his answers:

1. What do you regard as the lowest depth of misery?

Boredom.
It happened once in 1988, and was so bad I wrote a poem about it to remind myself to avoid it in the future:

. . . . .

I remember being bored
    once.
It was horrible.
Bored means
    Nothing
of
interest
at All.

It happened one month,
we were stranded for two weeks
    with no place to stay
(move, move).
We ran out of fun
    and energy and
thoughts.

Time
    we held still
for as long as we could,
but then we gave in and went home.

It lasted two days,
and was
    awful.

. . . . .

At the minute-to-minute level, though, the greatest misery is when I fall behind in some uninteresting assignment and have to slog my way for days or weeks of misery-per-moment to climb my way out. This occurred in school periodically, again when writing tests for a hardware subsystem for eight months straight, and when I saved all my annual bookkeeping and accounting 'till December (that's when listening to Gregorian chants for 12 hours a day saved my sanity (try the CD "Chant").)
==

2. Where would you like to live?

Boulder, Palo Alto, Salt Lake City in that order.
Salt Lake City isn't bad if you get out into the desert or mountains often enough, and leave town

periodically for a good conversation and pub crawl. I'm sure the same actually applies to Boulder and Palo Alto. (Palo Alto could stand to rediscover grass, though).

==

3. What is your idea of earthly happiness?
A deep and interesting conversation.
A good mid-morning nap.
Feeling the texture of sitting still, underwater.

==

4. To what faults do you feel most indulgent?
Sheer enjoyment, as the excuse for not doing something that was supposed to be done.

==

5. Who are your favorite heroes of fiction?
Spaceman Spiff, for imagination, daring, and post-failure rationalization :-).

Garfield, for sheer enjoyment of eating and sleeping.

The pig farmer who becomes king after a long set of adventures and then realizes he'd really rather be a pig farmer, but has to stay king.

Rick (from "Casablanca"), the reluctant - and lazy - sage and hero. (the pig farmer who figures out he'd really rather be a pig farmer, after a set of adventures but before he gets stuck being king, and so goes back to running his bar.)

==

6. Who are your favorite characters in history?
Socrates, for his indefatigable honesty in inquiry
(forgiving that he was also cantankerous)
Omar Khayyam, for choosing to live a quiet life of astronomy and poetry, and for his poems
are much better in Farsi than in Fitzgerald's English)
Musashi, for overcoming immense odds, using all tools available and inventively adapting strategy to situation, and for leaving us a nice book to read.
Benjamin Franklin, for his initiative, unorthodoxy, wit, and communication skills
(appreciating but forgiving his rationale for nominating the wild turkey as the national bird of the United States).
Mark Twain, for his keen observation and immense wit in writing
(commiserating with him for eventually sinking from sarcasm to defeated cynicsm)
Mohandes Ghandhi for illustrating aggressive passificsm, and showing outrageous belief in multitudes of people
(forgiving that he was also overbearing).

==

7. Who are your favorite heroines in real life?
Every housewife/mother who declines other income to raise the kids.

==

8. Who are your favorite heroines of fiction?
Laura Croft (see #15).
==

9. Your favorite painter?
Magritte, for technical quality, unconventional humor.
==

10. Your favorite musician?
Bob Marley.
Dave Brubeck.
(yes, I know that's two. I'll take both)
==

11. The quality you most admire in a man?
Compassion
==

12. The quality you most admire in a woman?
Compassion
==

13. Your favorite virtue?
Honesty. (see Socrates, Musashi, Ghandhi).
==

14. Your favorite occupation?
Writer.
==

15. Who would you have liked to be?
My current life vastly exceeds my prior imagination. (Please stop asking "Where would you like to be five years from now?" — it has never happened that my life five years later was nearly as boring as I would have requested in my answer to that question. I have no idea what path led me from anywhere to here.)
I haven't found a way to integrate complex physical activity with the mental and social; being an astronaut might have been such a one and would be a nice thing still to accomplish. On the other hand, it might turn out to be immensely boring. Maybe I'll find out.
==

16. Your most marked characteristic?
"Making the familiar strange, and the strange familiar."
For many people, particularly the former. For me, particularly the latter. My boss commented when I was 22: "If there is a known path from A to B, Alistair will find another." Don't hire me to do what you already know how to do.
==

17. What do you most value in your friends?
Honesty. (see #13)
==

18. What is your principle defect?
Procrastination. (see #1, leaving the year's accounting and bookkeeping until December).
==

19. What is your favorite occupation?

Conversing with thoughtful, educated people
(a tight game of squash is the physical equivalent).
==
20. What historical figures do you most despise?
Those who deliberately caused pain to others.
(there's a lot of competition for top ranking)
==
21. What natural gift would you most like to possess?
Patience.
==
22. How would you like to die?
Plane crash. Heart attack in my sleep and drowning in the North Sea are runners up, as I also wouldn't feel the end, but the plane crash pays double indemnity.
==
23. What is your present state of mind?
Torn between relaxing enough, and getting to everything I want to do.
(e.g., my hammock is out, but the mint juleps keep melting)
==
24. To what faults do you feel most indulgent?
(this is the same as #4 - Sheer enjoyment - I assume the duplication of question is an accident)
==
25. What is your motto?
"stasis is stagnation," coined at age 20.
(still holds)

Volume V, Issue 1

INTERVIEW WITH MARY POPPENDIECK
"ISSUES FACING THE AGILE COMMUNITY, PRESENT AND FUTURE"

By Nancy Van Schooenderwoert nancyv@agilerules.com

Co-Founder of Agile Rules www.agilerules.com

The following is an interview I did with Mary Poppendieck this month – April, 2004. Mary's background gives her a unique vantage point on the agile software movement. She worked as a programmer, building process control and vehicle control software on mini computers in assembler and Fortran early in her career, and later as a manager where she built one of 3M's first Just-in-Time lean production systems. She, and her husband Tom, wrote "Lean Software Development, An Agile Toolkit", the book that showed us how and why lean manufacturing principles underlie agile software practices.

NJV: What practices are truly essential for agile software development? There have been online discussions on how to define a minimal necessary set of practices for agile software development – what's your view?

MP: I missed the online discussion but I have some opinions. I think the essential thing is that the work people are trying to do has to be of value to the organization they're doing it for. They have to focus on delivering that value, not focus on other stuff. That's pretty much a Scrum thing – It forces people to think about the value being delivered.

Test-centered processes are fundamental. Whether it's test-first, automating testing, or using tests for requirements, especially for acceptance tests. Test centered processes are fundamental.

People-centered leadership is fundamental. Management focusing on the people doing the work, and empowering them, is fundamental.

I think the concept of very short batches of work – iterations or whatever, focused on delivering value, is fundamental. You have to have short delivery cycles.

NJV: In your book you talk about lean software development and you relate that to the lean manufacturing revolution. Are you an advocate of one of the existing agile software methodologies or do you propose your own, or some mix? What are you a proponent of?

MP: (Devilish laugh here) Good question. I have a hard time with the concept that there's any one best way to do anything. From a point of view of software development, I don't think there's any one methodology. I think there are principles that people can and will tailor to environments. But I just *cannot* force myself to get in front of any group and say 'this is the one best way to do things'. So in that sense I would say that I'm not a proponent of any particular methodology. I think that some of the methodologies out there have some very good ideas. I particularly like Scrum's focus on business value and on being people-centric. I particularly like XP's focus on testing and on real short iterations. If I were talking to somebody – I would try to figure out which practices of those [methodologies] fit that company's organization. Rather than be a proponent of *a* particular methodology, I prefer to be a proponent of a way of thinking about things that leads to some of these methodologies being used.

NJV: It sounds like you are saying to start with the principles behind the methodologies, and come up with practices that seem appropriate for any individual situation. Is that a fair statement?

MP: Yes, no situation will be a mimic of the next one, so no particular way of doing things is ever going to work everywhere. Software is just too big of a topic! There are too many ways to do software to have one and only one way to do it. I don't believe in this "one best way" – actually the quote 'one best way' is from the guy behind Scientific Management – Frederick Taylor. He is responsible for the idea that engineers can break work down into tiny little pieces, each piece can be optimized, and then the engineers can teach workers the 'one best way' to do things. The Toyota Production System is an antidote to Scientific Management. It says: no engineer is going is going to be able to figure out the 'one best way' to do things. Only workers can figure out the best way to do their jobs, and today's best way had better not be cast in concrete, because tomorrow an even better way must be found.

NJV: I'd like to get your view of the push for outsourcing and offshoring jobs in the software industry. Where is that going and how should people in the agile movement react to it?

MP: Well, I don't want to comment on how that's going because I suspect it's going differently in different organizations. Some organizations that are wise and have thought about outsourcing carefully are probably having good success with it and other organizations that are just kind-of winging it probably are not.

As far as how agile people should respond to it: The Agile movement needs to figure out how to focus on the kinds of development that will help companies have a significant impact on whatever business they're in.

One of the things that happened in the 1990's is that a lot of money was spent on IT but if you look at the actual delivered business value of these investments you do not see that that investment led to a lot of either increased revenue, or decreased cost. So a CIO in a moderately large company may be in a situation where a large investment has been made but not a lot of return has been seen. And they're going to be under pressure to spend less or deliver more value. So one of the things I'm noticing in the CIO world is a big focus on 'Let's stop just doing IT – let's start delivering real value'.

The second thing CIOs are looking for is fast delivery. And the third thing they're looking for is efficiency because they have huge server farms, and people tending those, software licenses, and all that kind of stuff, and they have to figure out how get the most out of the money being spent on that. Finally, they're worried about integrity – about security, and everything it takes to have reliable software. Those are their worries.

It is ok if there are certain efficiencies that can be gained from outsourcing. The primary thing CIO's have to do is deliver real bottom-line value. So they need to figure out what kinds of things they do are just going to keep the lights on and match best practices of competitors – survival type stuff. And then they need to figure out what's strategic – what's going to help their company thrive. If CIO's are smart and they're outsourcing then they're outsourcing the survival stuff. CIO's who outsource strategic stuff are probably making a mistake because they are giving away their secrets and competitive levers to other companies.

One CIO who claims he doesn't do any outsourcing is Randy Mott, the CIO of Dell, formerly CIO of Wal Mart. Here's a guy who's headed up the IT area for two of the world's most successful companies and he basically says 'Look if I outsource then there's an intermediary there, and I can't believe that I don't have to pay for that intermediary. And it seems to me that I should be able in my company to focus everybody I have on the stuff that's going to differentiate my company and allow it to grow.'

NJV: Suppose you were the CIO in a large company and your management was pressuring you to outsource. You've already mentioned that you'd be looking at "survival" type things for that. Could you elaborate a little further?

MP: I'll go back to the example of Randy Mott on this one. One of the things Mott says is if you look at your operations you'll find that on the average 70 to 85 percent of IT resources are being used just keeping the lights on. This means running the servers and patching things, doing security, and nurse-maiding legacy systems and things like that. He says that's not where you want to be spending your resources. Now, what I observe is some people take a look at this 70 – 85 percent for keeping the lights on and say 'Gee I could keep the lights on cheaper if I had cheaper labor' and that's probably true. What Randy Mott says instead is 'You know what – that's no way to be spending my resources. I have got to figure out how to spend less of my time on things that are not strategic.' So what he does instead is try to build his way out of legacy, automate things that can be automated, have a very uniform infrastructure across the company that can be rapidly built into. He requires aggressive integration of everything that's done all around the world. His idea is 'I have to spend less time and be much more efficient at keeping the lights on'. Now some people are going to say 'Ok, to be more efficient I'm going to go offshore'. He says 'To be more efficient I'm going to figure out how to do it with less effort.'

Now if I go back to my manufacturing days, one of the mistakes that we made – I even made – was we would automate a factory before we figured out the best way to operate it. So we'd put in material requirements planning scheduling stuff and we thought that that would help because it would provide more information. But what it did was lock that plant into what was a fundamentally bad way to run the plant. Now one of the problems you have if you start outsourcing – you have all of this 80 or so percent of your resources being used to just keep the lights on, and if you start outsourcing that just to drive down the costs then where are you spending the time asking 'why am I doing this in the first place?'

The more important thing, before you outsource stuff is to figure out how you can get rid of doing all that junk. Mott asks himself 'how come I'm spending all this time supporting legacy systems – I don't want to have legacy systems'. He routinely advises CIOs 'build your way out of legacy – there's nothing good about it'. The concept of outsourcing first, before you figure out whether or not you want to be doing it can definitely be a mistake.

First of all you don't want to outsource strategic things. Second of all you want to get your operational act together before you outsource. Well, people are using outsourcing to *get* their operational act together. That might be ok but I think they may be casting in concrete stuff they don't want to be doing in the first place.

NJV: I'd like to ask you about cases where agile methods have not been successful. An example is criticisms made of XP in the book "Extreme Programming Refactored" – any comments?

MP: Can you remind me – what was their most important criticism?

NJV: One was that XP's practices are so interrelated that unless all are followed correctly then the whole effort collapses like a house of cards; that XP is a fragile, even unstable process for this reason. Additionally, I've seen in some narratives online where people begin applying XP and they get resistance (it varies case by case of course), then things seem to unravel.

MP: Ok, I have two answers for that. To the first point – one of the criticisms I have of XP is that there are too many people who say 'I don't know why it works so just do it all and you'll figure it out'. Well, what if you do it all and you don't figure it out? There are underlying principles that make XP work and that's some of what I was trying to expound in my book – what those principles are. If people just do practices, and they do them in a different environment without understanding howcome various things are important, then there are definitely going to be failures. In fact I have seen no good idea come along *ever* in all of my years that hasn't been misapplied frequently. Whether it's lean manufacturing or anything. Every one of those ideas - when applied as rote "do these mechanisms" rather than fundamental ideas of changing the way people think and the way people lead and the way people treat each other – had many failures. In fact anything that becomes popular will be copied badly and will therefore lead to failures. You can't copy rote practices; you have to understand the principles and apply them to your environment.

The second point, about resistance – I happen to believe that fundamental change in the way things work in an organization is going to have a very difficult time from the bottom up. It has to come from the top down. We need to elicit support from the senior management.

Let's take a case where they try XP in one project over here, and the XP works well. Ok now let's take a look at the rest of that organization and how do they feel? They feel that their dearly held beliefs are being challenged. The other people not involved in this thing [the XP project] have no incentive to find it good, because it's threatening the kinds of things they believe in and the way that they've been doing things. People talk about organizational antibodies coming out and I believe that's a natural event. If you try to change the underlying paradigm in an organization from the bottom, your likelihood of success is low. What we need to bring agile ideas to a much larger group of people is we have to get to a CIO level and make sure that those folks understand the important benefits of what this can provide them.

NJV: Suppose a manager wanted to introduce Agile in a pilot sort of way – is it inevitable that they'll run into this sort of organizational resistance?

MP: If a manager wants to do something their best bet is to get colleagues at their level, and their immediate boss, rooting for them. What they need to do is create a network of people not involved in the experiment who understand what they're trying to do and would like to see it successful. They need to be selling the concept much more broadly than with just their individual team. Otherwise everybody else is just going to feel left out. And why would people who are left out of the picture think it's a good idea?

At 3M when we had new ideas we wanted to propagate among people, we'd have reading groups. We would study the theory behind why things worked. We'd have forums and talks. There's a person who works at a large company nearby who is trying to spread the concept of agile in his organization which does software for hardware devices. He is very good at bringing in speakers and getting teams together to think about this and talk about it over lunch. He tries to spread the idea sideways in the organization. He doesn't try to have one example and say 'Look you dummies why don't you do it this way'. You need an approach that's inclusive of the organization, rather than exclusive of it. In fact, my bias is not for project retrospectives, but for department-wide studying of how projects work and can be improved. There needs to be a way to spread the ideas from individual projects across similar areas of the organization. I'd like to see organization-wide mechanisms to get ideas to people.

An organization isn't going to change with a small one-project team: it's going to change through an infiltration crosswise of knowledge, and I don't see that emphasized in the agile movement. I don't see us addressing organizational issues; we tend to focus on one project at a time. One project at a time isn't going to cut it. Real change happens on a broader basis. One favored team at a time is going to create resistance. It's guaranteed – it's so basic to human nature that you can just about write the end of the story before the experiment takes place.

NJV: You mentioned that change has to be brought in at the CIO level – is that high enough?

MP: The level depends on the organization and the situation. Let's go back to the company I mentioned above. The agile champion has been very effective at getting the agile ideas across his teams in the software division but he's having a difficult time in firmware and hardware development. But they need to be integrated with the firmware and hardware people in order to make this work. Now he needs to move up and broader, sideways – that's where he needs somebody higher than him to get involved. It just depends on where your borders are. Start where you are, but the higher you are, the more you can influence.

NJV: Imagine you're heading up a software team to build a very large-scale life-critical system – let's say it's the next manned mission to Mars. How will you go about getting the right practices to happen?

MP: I'll answer that by cheating and telling the story. There is a book called "Deadline" which talks about projects that succeed against aggressive deadlines and what caused them to be successful. A project of the approximate size you're talking about is covered in the book; it's the Boeing 777. Boeing promised to develop and deliver the 777 to United in a timeframe which was very, very aggressive.

Now, start out with the fact that Boeing knows how to run projects – knows how to do high level systems engineering – knows how to drive schedules – knows how to work with vendors and put planes together. The unique thing about this particular project was that it had to be done faster than they had any right to believe they could unless every single thing went right. They added another layer on top of all these underlying good disciplines. If you don't have the underlying disciplines, it's curtains. You've got to have that. But they did have it. So what they added on top of it was a program called "Working Together". My husband was working at Honeywell at the time and they did the gyroscopes and even he was touched by this "Working Together" program. It was a mechanism of forcing early, detailed communication between the various people on the team, from the mechanics who were going to maintain the plane all the way to the people doing the systems design; moving information among vendors and the customer and getting high involvement of that whole span of people very early. There was a lot of early release of information and examination of it. In fact they did deliver on time, and it's largely credited to this very aggressive cross-functional communication program.

For an example, there was somebody early in the development stage who happened to be at a review of the design who also happened to know a lot about fueling airplanes. And he said "You know what – there isn't a fuel truck in our fleet that has a hose long enough to reach that spot where you're expecting us to put fuel into this plane." And no one else had thought of that. If that person hadn't made that comment at that time, they speculate that they would have delivered the plane with a fueling port no fuel truck could reach.

My answer to your question is this: if you have a large, complex project you need two things; you need excellent communication flow on top of a disciplined environment that already knows what to do.

I don't think you can do anything complex without screwing it up. So the more eyes you have on it, figuring out what could go wrong at the detail level – not the overview – like "can I really fill this fuel tank" – the more eyes you have of people with experience in all those different areas, looking at the design earlier, the more likely you are not to screw up. You need early, frequent feedback loops from the people who know what can go wrong. You've got to create that environment.

NJV: I've got a 3-part question for you concerning the future of agile.

Where do you think the agile movement is headed next?

What is the main present threat to it?

What do you see as the next challenges in software development and how can agile help?

MP: I would like to see Agile cross the chasm and start becoming commonly used in medium to large sized organizations as the natural, accepted way things are done. If it's going to get there, we need to start talking the CIO's language and I think there's a danger right now – Agile may be perceived of as a reaction to someone else's problem. If we worry about big design up front and say it's terrible, people who have a problem with no design rather than big design will not think we are talking to them. We may have to abandon the things that sort of spawned the movement if you know what I mean.

NJV: Can you elaborate a little further?

MP: To some extent agile is seen as a reaction to over-designing and over-planning and over-emphasis on technology rather than business value. Now the problem I have is I'll go into an organization that is chaotic and agile isn't seen as a cure for their problems. I was talking with an executive who said 95% of his colleagues don't know about agile and he'd put about half of them in the waterfall area and half of them in the chaotic area. We've got two constituencies we need to talk to – those that have over-control and those that have under-control. We have a very good story for both of those but if we are perceived as a reaction to over-control we are not going to talk effectively to those that are in the under-control area.

There's a danger of the pendulum swinging too far. I think it would be wise for agile to stop worrying about what agile is against and start trying to put on the head-set of the CIO and the mindset of the senior people and figure out what they are looking for right now, and how can what we do address those issues. That's what I've been working on recently.

One of the things I see CIOs having to think about right now is value-based portfolio management, where the business value is owned by the business organization, and fast delivery, efficient operation and integrity. We have a lot to say about that. We can talk about test-centric processes, people-centric leadership, the fact that by dividing work into very small iterations we create one of the most efficient mechanisms to move work through organizations.

We've got a lot to say to a CIO but we're not delivering our message in the terms that they want to hear. We're not telling them how it's going to solve their problems – we're telling them how it's going to solve developers' problems. We've got to start worrying about management problems and how what we have here can solve those problems. Because otherwise you aren't going to get that leadership push to make this acceptable in organizations.

We've got to stop giving the impression that agile is a document-less, free-wheeling, very rapid, no-planning kind of an environment. Instead we have to say agile is the way to make sure you get business value. This is the way to get rapid, high integrity delivery, this is the way to get much more efficiency out of development. This is the way to make sure that you're delivering the strategic stuff – the 20 percent that's going to deliver 80 percent of the value. This is the way to re-think your performance measures and measure the right things. If we can sell agile as an excellent tool for those kinds of things – and I think it is – then we'll be talking the language that managers we want on our side need to hear.

There is a lot of agile development going on, but it may not be known by the word 'agile'. If you take a look at Dell, for instance, Randy Mott has listed his objectives for IT, and they are

* Projects may take no more than six months

* Every project must deliver measurable business value

* Every project must be on time

* All software must be tightly integrated

*All software must be usable globally

Mott lists what people need to be trained on:

* First and foremost, people in IT need to know the business.

* Secondly, they need to understand what leadership really is.

* Thirdly, down at the bottom, they need to learn technical things.

All of Mott's ideas fit extremely well in the agile movement. But it's not called "agile", just common business sense.

I think common business sense is going to win out in the economic world. In that sense, the stuff underlying agile will prove to be the stuff that's successful. Whether the agile movement ends up being successful, I don't know. But projects are going to get shorter, much shorter. Business value is going to be the key thing that people go after, and the successful companies will be the ones who make sure the developers really understand the business, are working in a correctly disciplined environment, and are led by leaders that empower rather than tell them what to do.

Books mentioned are:

"Deadline!: How Premier Organizations Win the Race Against Time" by Dan Carrison

"Extreme Programming Refactored: The Case Against XP" by Matt Stephens and Doug Rosenberg

"Lean Software Development: An Agile Toolkit" by Mary and Tom Poppendieck

THE AGILE SOFTWARE CODE FACTORY, A NEW WAY TO LOOK AT THE ORGANIZATION

By Cliff Gregory

Recently, it was necessary for me to look closely at how my company is organized. We have been using agile development methods from the start, but the Organizational Chart looked and felt like a traditional company. As a matter of fact, it reminded me a great deal of any hierarchical organization, including the military. I was feeling that we were not getting the best from our "Agile" efforts. Considering our dedication to being an agile company, and the fact that many traditional roles are used differently when applied agilely this view seemed inadequate.

First, it seems that even calling us a "software development company" is slightly inaccurate. We are making computer software to support enterprise installations, true enough, but it is being created in a vastly different manner than it been in the past.

Manufacturing can be defined as the act of producing goods from raw materials melding both creativity and technology. Since we build software from the raw materials used in our industry, like blocks of coded language and mathematical formulas using technology tools such as SDKs, IDEs, defect tracking and code management tools, I believe we should use the term software manufacturing company to be more accurate. This association leads to the recognition of the people building software in a software manufacturing plant as a new sort of "factory" worker. Like the early auto factory, we are finding agile means to manage leading to greater production without a larger commitment of creative resources. This is not a term that everyone would chose, but I think that you have to look below the surface to see the reasons, and there the term makes sense. Factory workers were the first real skilled, creative workers in the world. Creation of goods for sale in a free market is one of the most important tasks in a civilized world. The ability to product high quality goods consistently is one of the true driving forces required for automation to work. It is automation that enables civilization.

Historically, development groups have been organized into corporate structure, working in a loose association instead of a coordinated effort of very creative people working toward a common goal. I have always preferred to call these groups a "team" to better denote the level of commitment and commonality of purpose needed to build software successfully, but even that term falls short of the level I am trying to express.

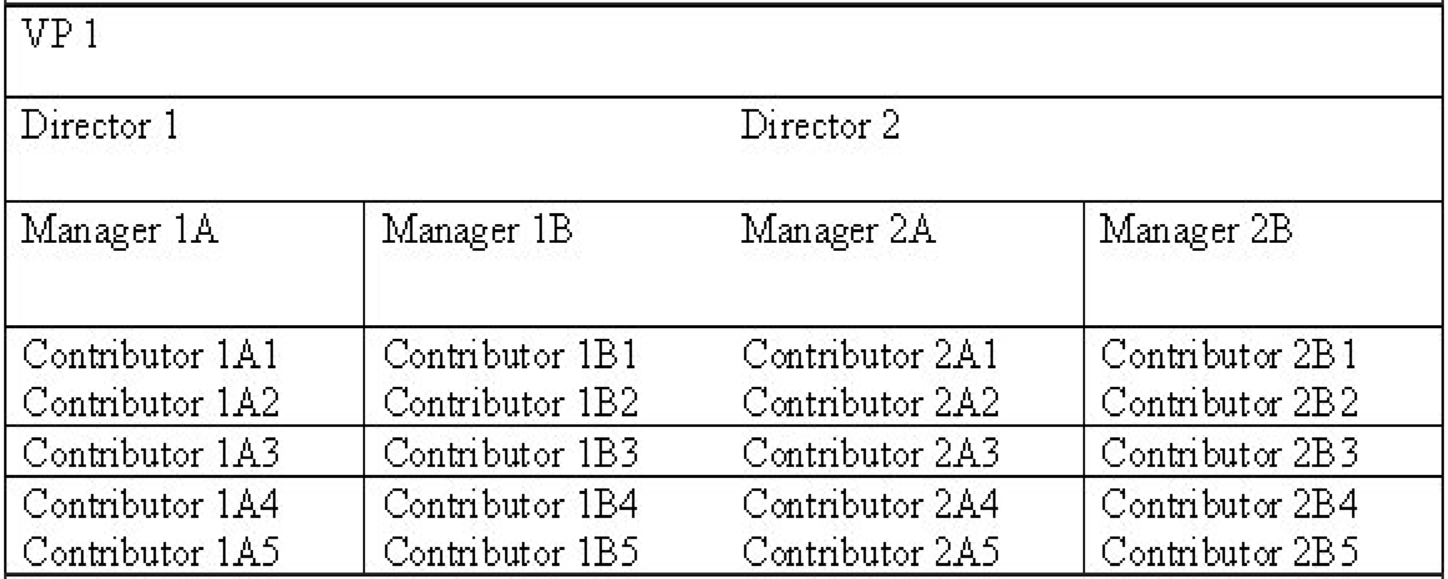
I recently read part of a presentation by Luke Hohmann (Beyond Software Architecture – Addison Wesley, 2004). Luke referred to development workgroups as Clusters, and I propose that using the term cluster suggest equality of interaction and commonality task. In an agile organization, keeping the focus on people over process and product over documentation has opened the need for better and more effective interaction and commonality of development teams. Using "clusters" allows for a functional view of a traditionally hierarchical organization, and expresses the oneness of purpose while holding each individual's creative personality. Applying the concept of a cluster to a code factory opens the door to free communication, and it permits multiple levels of relationship between clusters especially as they are associated parts of a larger or smaller clusters.

As it is currently viewed, a company is organized into work groups, loosely based on the function of the group leaders. As an example, the CEO has a group of direct reports, VPs and C Level managers, who dependent on their function will have a group of reports at the VP and Director levels and so on until you reach the individual contributors. If any individual contributor within a discrete group needs to communicate with an individual contributor in another group the path of communication is expected to follow along organizational lines including the managers in each line. Often, these lines are the basis for small skirmishes between managers trying to protect the integrity of their group. This entire process is counter to productive behavior, and brings individual manager's egos into conflict. This is not a way to reach agile levels of productivity and non-interference with creative assets.

In "clustered" groups communication passes through "communication junctions" that may not have ownership or direct influence on the individuals trying to communicate. Lines of functional leadership, pass to groups, who then serve to facilitate effective responses and short-circuit road blocks as they happen. The ago old problem of finding the single person who is preventing effective communication becomes a thing of the past. All lines of communication are functionally able to seek multiple paths, leading to groups and individuals being able to find their own best method. In short, much like water, communication can seek the path of least resistance.

Formal approval, of course must be maintained within a marginally hierarchal structure, but if the organization can force decisions to the lowest possible level, and allow managers to develop adequate information bases they will enable informed choices. The approval process can be shared among managers within a cluster without losing overall control. The key is to have open communication, eliminate competition, let people and creativity drive the project rather then the desire to win the race.

Figure 1 below shows a standard view of an organization chart, without identified roles, which has inherited communication roadblocks and often disables effective interaction between individuals and groups.

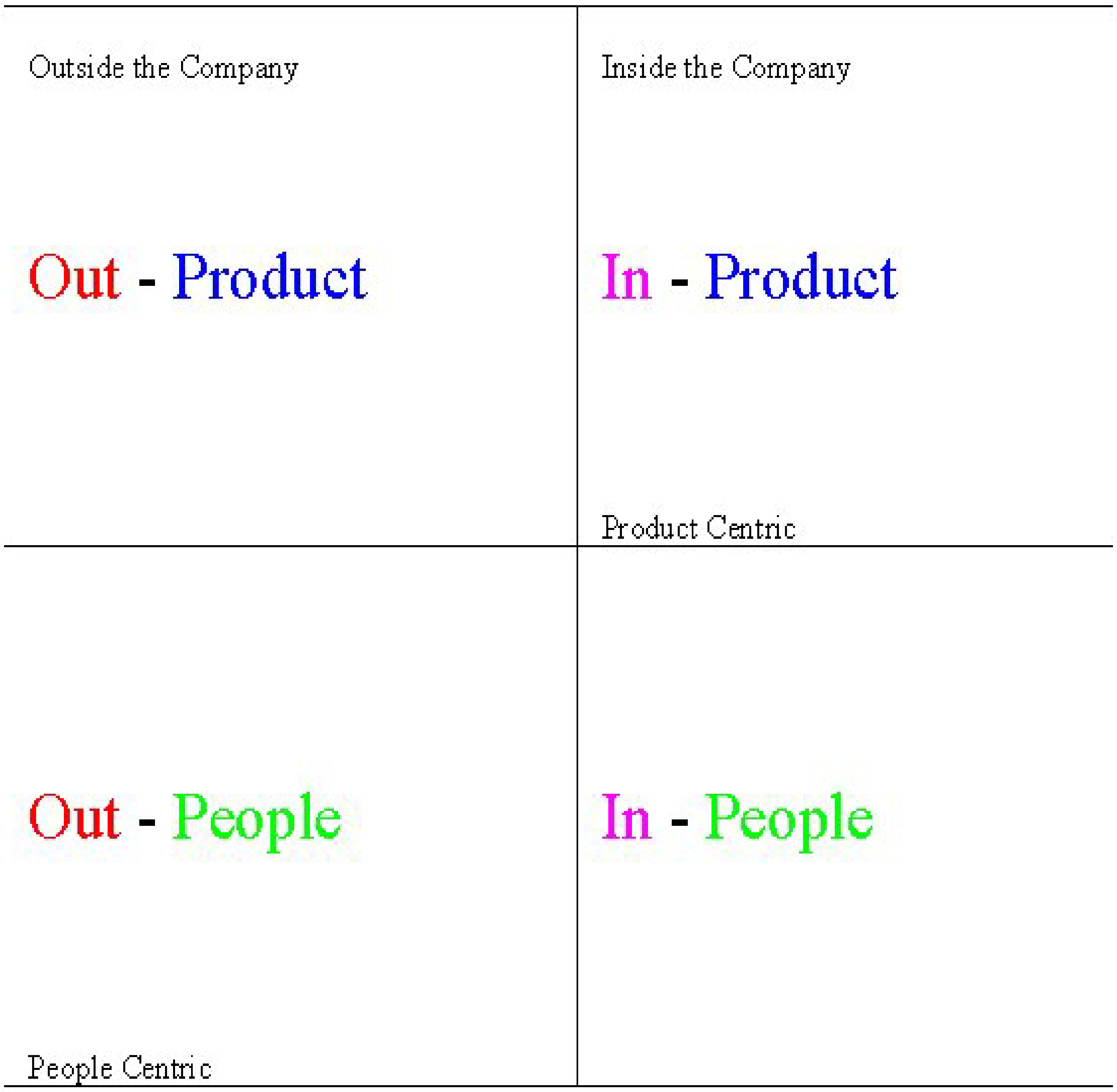


VP 1			
Director 1		Director 2	
Manager 1A	Manager 1B	Manager 2A	Manager 2B
Contributor 1A1	Contributor 1B1	Contributor 2A1	Contributor 2B1
Contributor 1A2	Contributor 1B2	Contributor 2A2	Contributor 2B2
Contributor 1A3	Contributor 1B3	Contributor 2A3	Contributor 2B3
Contributor 1A4	Contributor 1B4	Contributor 2A4	Contributor 2B4
Contributor 1A5	Contributor 1B5	Contributor 2A5	Contributor 2B5

When a Contributor working for Manager 1A under Director 1 wants to deal with helping a Contributor working for Manager 2B under Director 2 he may require permission from five separate managers. If any one of them is disinclined to support the interaction, they can block the efforts. This structure promotes communications failures and short circuit management's positive efforts to interact effectively. Additionally, if anyone in the chain is unavailable, it slows response until they are reachable.

By clustering groups and managers across organizational boundaries into teams with like focus, you can create a secondary path to communicate and reduce the number of "turf" battles that arise between groups. Managers and worker who have unlike organizations and functional areas, but have like focus on things like customers and technology present an informal method to communicate. By assigning a number of "communications junctions" along these paths, a team can effectively double or triple the paths to effect interaction within and among teams. See figure 2 below to start the process.

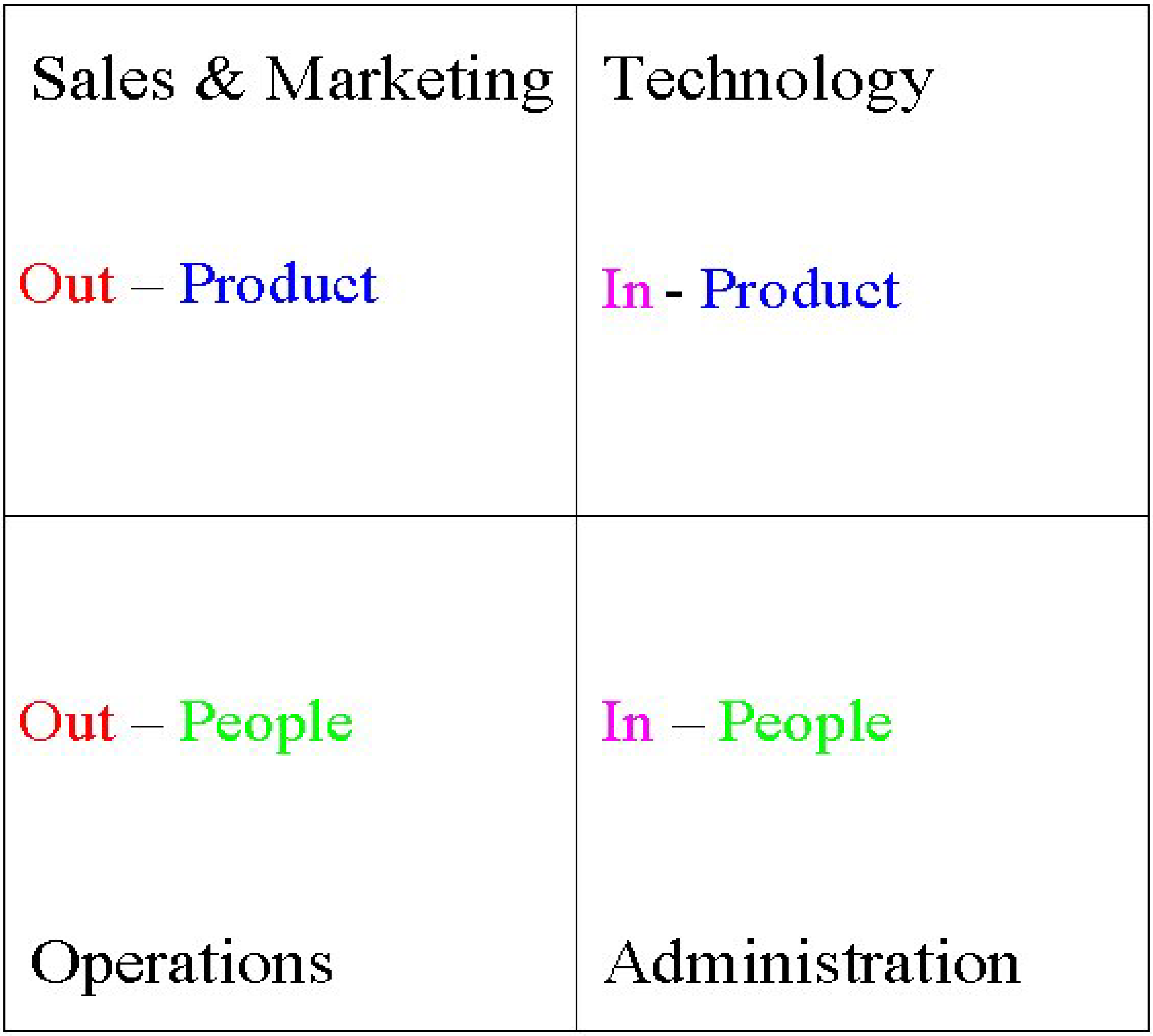
Figure 2



This method allows a separate window into the dynamics of groups working for a common goal. I choose to refer to these two types of groups as Organizational Clusters (Traditional Organizational Chart Groupings) and Focus Clusters (based on the focus of each individual within each group.

At the highest levels within a company, it is uncomplicated to divide by focus, but at lower levels it can get harder. As an example, I will group a company into this method. Remember it does not replace the hierarchical method, but augments it. The organizational group is identified by focus rather function. Figure 3 shows highest levels of a company functionally separated by focus.

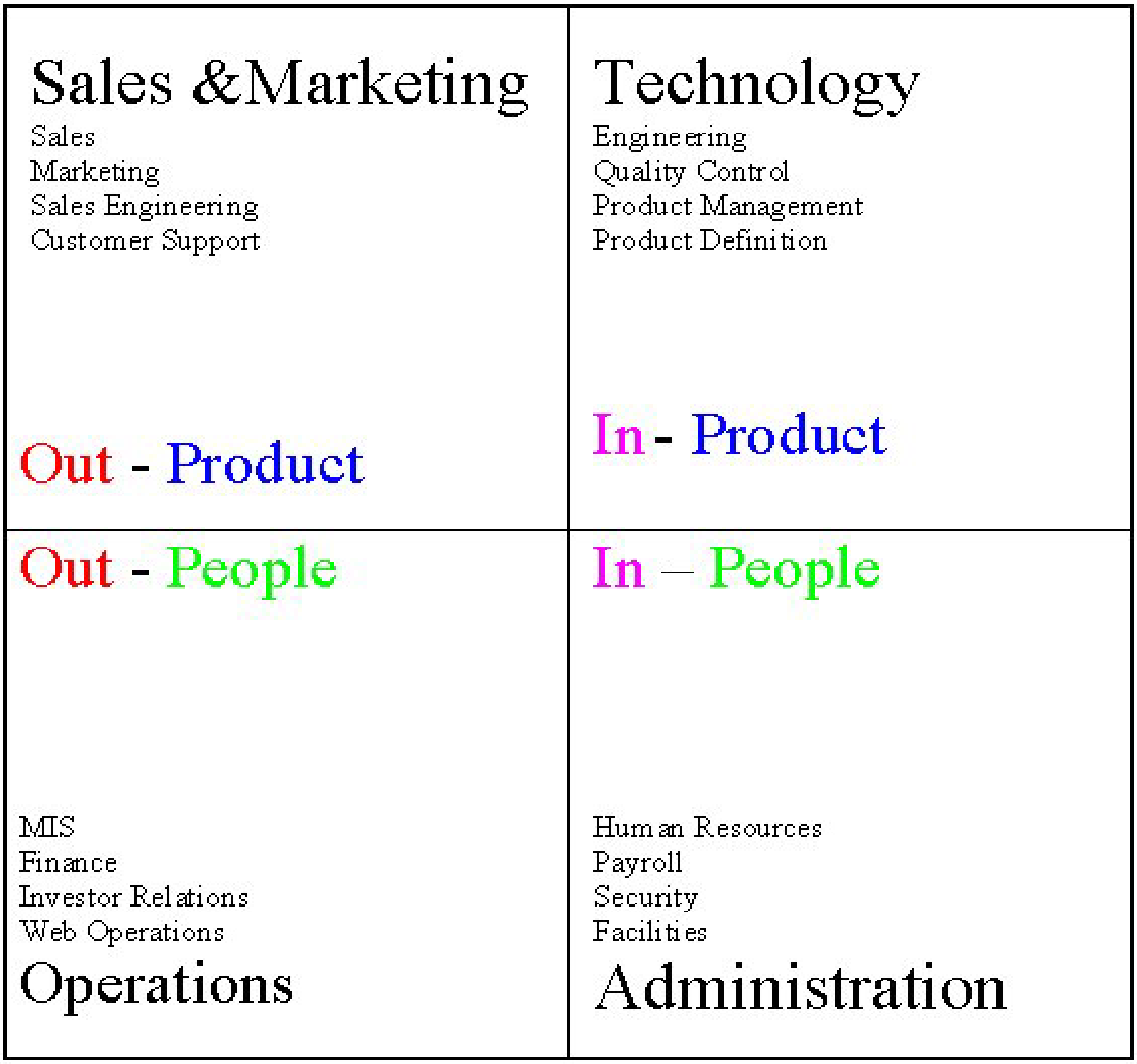
Figure 3.



When the basic functional/organizational units are identified this way, the relationship between roles is easy to understand. In Figure 4, I include additional functions.

Some could be included in two or more areas of the graph, but for the purpose of keeping the separations clear, I placed them into a single focus area. If a function is listed here it may not be the way you would list them, but for this paper, I will use them in this manner. You may structure your group as you and your team view the focus shifting across organizational boundaries. I only suggest that you work with the entire management team to attain a structure that you can all support. At this point the way you divide the organization is not the most critical, in fact, the importance is to attain a division different in focus from your organizational chart roles.

Figure 4.



Finally, figure 5 below shows a complete organization, divided into focused groups setting the stage to build clusters at the top level of the organization. Each cluster is built by linking across rows so that Operations/Administration Cluster is related to the Operations/Sales & Marketing Cluster, is related to the Sales& Marketing/Technology Cluster, which in turn, relates to the Technology/Administration Cluster. The Operations/Technology Cluster links to the Sales & Marketing/Administration Clusters. The points where each cluster links to another cluster are the points of communication I referred to in an earlier paragraph as junctions. They enable communication between peers in various groups without the involvement of managers. The last refinement showing how a Focus Clustered Organization communicates resembles a Pyramid with each focus area holding equality of voice with peers on their level, and hierarchal relationship within their Organizational Cluster. This enables multiple paths for effective communication.

Figure 5

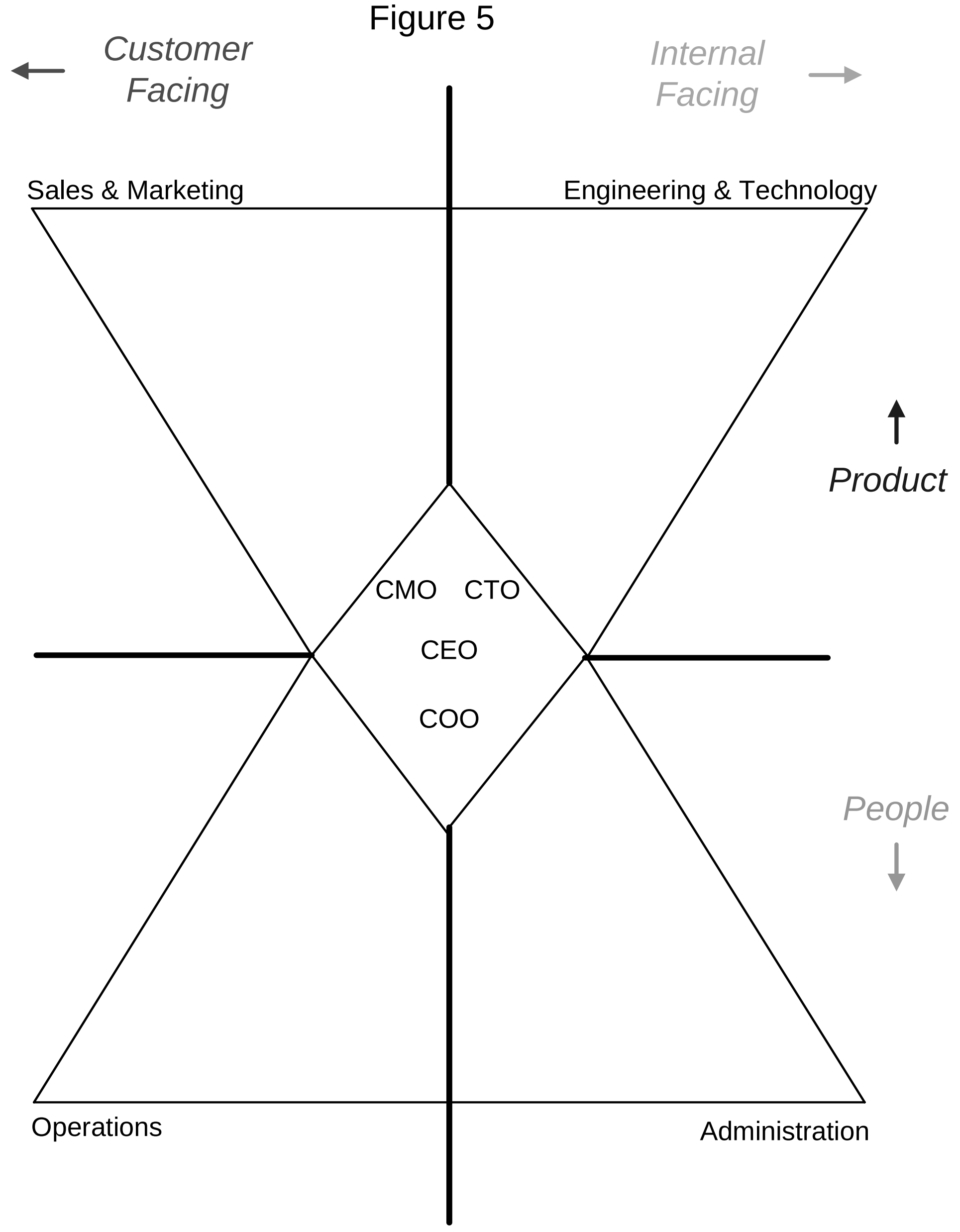
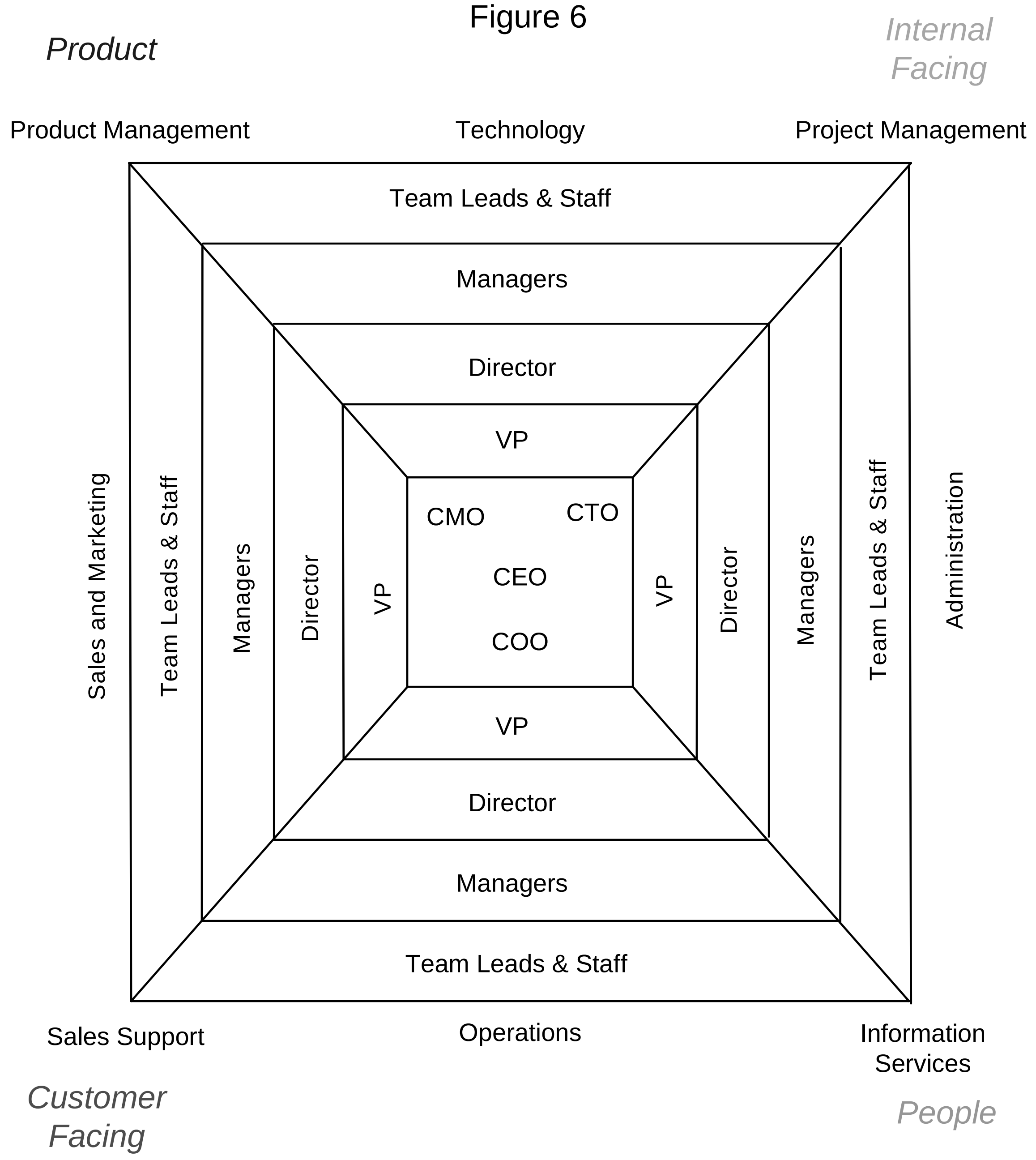


Figure 6, below, shows those Focus Clusters connected to the more standard organization structure, to give you many paths to communicate. This entire structure works to help manage already accepted informal line of communication and give them some legitimacy, but it comes at a cost! When lines of communication are increased, so are lines of responsibility and the need for solid, objective, metric-based data to give a universal picture to the entire organization. While, traditional development and reporting

methods could certainly provide this data, I believe that Agile development groups need to build a better means to gather data objectively without impacting the creative resources.

Figure 6



In placing people over process and product over documentation we seek to limit the impact on creativity historically caused by these traditional development methods. Metric-based, data collection from third-party control sources may be a big piece of this process, and allow your company to build an organization built with teams of formal structures and informal clusters.

Sharing actions taken and reasons for those choices with all levels of the team becomes an important function for the formal and informal paths to assume. It raises the level of trust required among managers and peers as well as the pushing the authority to act independently to the lowest possible level. These must be considered and determined at the onset of using this structure, but it will increase production and relieve choke points in the current workgroups.

The concept illustrated in the figures above has been applied to an entire organization, but much more importantly, it can be scaled down to fit every size workgroup. By taking the function of each member and determining the focus (either External or Internal Facing and Product or People Centric, as in my example or whatever focus you feel best represents your organizational goals) and assigning peers to communications paths across organizational boundaries, you can create this type of communications flow chart to support your overall best function. My structure works well for my company, and gets it basis in our business plan, so I have used it as an example.

To summarize, I see this new way of viewing at an organization as enabling Agile Software Development by allowing optimal communication and pushing authority to act to its lowest level. To the credo of "People over Process & Product over Documentation" I would like to add "Communication over Organization".

Responsible adults work at things they feel a strong passion to accomplish. The result is almost every worker in a Code Factory will do the best possible work if simply given the opportunity to do it.

Mark Friedman prepared some figures in this paper. I wish to express my thanks for his efforts.

Introducing Agile to NonAgile Environments

Janet Gregoryr

What can you do when you want to move from a traditional environment to an agile one, and make the change a success? I believe there are two main areas in addition to the agile methodology itself that should be considered before introducing agile development practices into an organization. They are:

1. The organizational culture
2. The application

These subjects are especially important when you are supporting an existing application with a well-established development and release process. The first section covers the cultural impact you should be aware of, and the second section is about the legacy application itself and the affects that it has on an agile implementation.

ORGANIZATIONAL CULTURE

Too many times organizational culture is not considered when trying to adopt an agile methodology, and we wonder why it didn't work as promised. In an established organization, processes may have been around for a long time and won't change easily, especially if individuals have a stake in them. Each functional group has developed processes that meet their needs, and probably feel comfortable with them. Fear is a very powerful emotion and if not addressed, can jeopardize the change. If team members feel that a new 'agile' process threatens their job, they may not want to proceed.

All stakeholders need to understand the benefits of using agile methods for the development cycle so that they buy into the process. Change management practices can be used to help solve some of these people issues. Some of the stakeholders that need to be considered are:

- Upper Management
- Business Unit Managers
- Project Managers
- Business Analysts
- Development Team
- Quality Assurance Team
- Technical Writers

Each of these stakeholders needs to understand the impact that this change will have on their workflow. The following section talks about each of the roles and some of the issues you need to consider. The final section talks about change management and the role it plays in addressing cultural changes.

Upper Management

Upper Management wants to know the benefits in dollar and cents. The questions they want answered are:

- What business risks are you trying to mitigate?
- What is the business value?
- What is the Return on Investment?

Here's one example of business value: Offering the business a chance to get valuable functionality delivered sooner, and that functionality may generate revenue that pays for the remainder of the project.

In my current organization, we used a customer audit as a stepping-stone to introduce the idea. The audit pointed out many of our weaknesses that XP addressed. They agreed it was worth the effort since the previous process had not produced the desired result.

We all know that if you have support of the Upper Management, the change has a much better chance of success.

Business Unit Managers

Business Unit Managers want to know how agile development practices will help them gain control of their project. Some of the major risks of a project can be scope creep, incomplete requirements, cost overruns, and lack of quality.

The Extreme Programming practice of Customer involvement throughout the development cycle addresses some of those high-risk items. The customer determines what requirements get completed first, and ensures that what is built is what is expected. The only undelivered requirements are those that the customer chooses as low priority. This helps to reduce project uncertainty.

Sometimes development groups try to introduce agile 'by stealth'. This can work for small projects in groups that work independently. But if management hasn't bought into the process, there is always the risk that schedules are still handed from above. Contracts are set at upper level and the development team is told what features need to be delivered. The change needs to happen throughout the organization.

Once the management is convinced to try an agile process, the fun just begins. Organizations usually think of the development group first, but the bigger change is getting the teams that interact with the development group to buy in. It is not often that the development team works in a vacuum. Here are some of the 'people' challenges that you need to address when trying to change existing processes.

Project Managers

Project Management Offices provide a very specific defined role in traditional methodologies. In an agile environment, Gantt charts no longer serve the same purpose – some people don't believe they serve any purpose. However, I think used as a high-level planning tool to show release type milestones, Gantt charts can be used for communicating with the clients.

Iterations tend to manage themselves and the work done is visible to all. There is less controlling and more monitoring. Project managers can still be useful in an agile project because they can help the team stay focused. They can also work with the business side to plan future releases, and take care of many details such as release planning.

Business Analysts

Business analysts may no longer have a role since customers work directly with the development team. However, I have worked on XP projects, and know of others, which included business analysts on the team. There is no hard and fast rule.

The key point is to investigate what roles may be required in the new process. For example, if you do not have the luxury of having a customer on-site, the business analyst may be the one to develop the 'backlog' of stories for future iterations and releases. They may be the one to help the customer define acceptance tests or assist in a product management role. If you have multiple customers, you may wish to funnel all requirements through a single source and the Business Analyst becomes the customer advocate.

Development Team

Introduction of the development practices into the team may be easy but it may not be. There are many challenges and the success depends on the group of developers. Assuming the team has been together working on an application for a long time, habits have been formed, and productivity has probably leveled out.

> I have helped to introduce agile methodologies into several companies and find the biggest initial payoff is to open communications. The daily stand-up forces people to interact and to talk about what they are working on.

When you start making changes, developers can feel threatened. Not all developers have equal productivity and less productive developers can feel exposed by agile development. Pairing can be threatening as well if the members have not been trained correctly. Sometimes you have to make tough decisions and get rid of people who don't want to try the new way. A smaller team with productive developers can be more productive than a large one with less effective developers.

> *I have seen very successful introduction of agile methods into a small team where at least one member has been on an agile team before. It was enough experience to help infect the other developers*

Quality Assurance Teams

Testers who have been working in a traditional setting may have a hard time adjusting to a new role. If they have come from an organization that has an adversarial culture between development and QA, it may be difficult to change from being the afterthought to being an integral part of the team.

We worry about developers who can't adapt – what about testers who are used to building test scripts according to a requirements document? Can they change to learn to ask the questions as the code is being built? Testers who don't change their approach to testing have a hard time working closely with the developers.

The testers may be used to doing only manual GUI testing and not understand the automated approach. To help them adjust, you may need to bring in someone new who has worked on an agile team as a mentor. You need a lot of courage to face this problem.

To quote Jonathon Kohl in a recent user group session, "the common thread is collaboration, and testers supporting the developers through collaboration and feedback. The testers do whatever they can to help the developers by providing them continuous feedback on their work, as well as expertise to ensure things are testable. The testers can also help facilitate communication between the business and the developers, which in turn helps enhance that feedback channel.

Technical Writers

Technical writers become an integral part of the team as well. They can play a very important role in legacy system development since most legacy systems also have legacy documentation.

I have worked closely with technical writers who were part of the agile team. They played a very important role helping to document the system, but also identified areas of the system that needed more work. Technical writers approach the system from the end users point of view and can work very closely with the test team. I worked with one incredibly talented writer and if she researched an area first, she shared her findings with me so I could develop regression tests. In the areas that I researched first, she used my tests to help her write about the application feature. It was almost a symbiotic relationship.

Customers

Customers are expected to play a very active role in an agile methodology. They may need to get used to working directly with the development team after years of being segregated and not allowed to talk to them. I talk more about customers later in conjunction with problems associated with a legacy application.

Change Management

When implementing any change you need to be aware of the effects. The first stage can be chaos, where your team isn't sure what the new processes are, some groups are loyal to old ways, and some people are unsure and disruptive. People mistake this chaos stage for the new status quo. To avoid this, explain the change model up front and set expectations. Find the areas of the most pain and determine what practices will solve the problem. Accept and expect perceived chaos as you implement agile processes.

When you start iterative development, use retrospectives to provide people a place to talk about their fears and give feedback. Let people know that it's normal to be fearful. Be open; teach them it is acceptable to say they are fearful or uncomfortable. Discuss it, learn, make decisions and move on. Fear is a common response to change. Forcing people to do something they don't want is detrimental. Lead by example. You will produce better code and form better relationships.

A critical success factor is that the team must take ownership and have the ability to customize their approach.

WORKING WITH A LEGACY APPLICATION

Legacy systems can vary in size, complexity, language, audience, etc. How you adapt your agile methods to fit with your system depends on the variations. For example, writing unit tests in JUnit is easy when your system is in Java, but you need to be creative if your system is in COBOL. Automating your system level tests pose the same challenge if there has been none. Some of the problems you need to deal with are:

- Large code base
- Defects
- Documentation
- Existing clients
- Regression testing

Large Code Base

A large code base is one problem with a legacy system, and usually means that major refactoring needs to be done. It also typically means there are no unit tests in place and very likely no automated acceptance tests. How do you get to a stable code base with unit tests and full suite of automated acceptance / system tests?

I have had the opportunity to work with 2 different legacy systems; one was written in Power-Builder, and one in Java. It was difficult to find developers who had agile experience who had worked in Power-Builder, but the developers we hired were eager to try. They researched a bit and found that they could write unit tests in P-Unit, not quite as easy as JUnit, but it served the purpose. However, the system did not lend itself to an easy implementation of system automation.

The system written in Java was much easier to tackle. We took the same approach as with the first, but the developers already had experience with JUnit, even if they hadn't tried TDD. Each new feature story had unit tests written. For every refactoring story, unit tests were written first to ensure that functionality did not change. Over time, unit tests will be written for every piece of code.

Defects

Another problem experienced with legacy systems is the number of bugs that exist. The theory of not keeping a bug database is out of the question as you can find them easily, and they can number in the hundreds. If you don't track them, you get used to looking at them and don't even realize they are bugs or inconsistencies. There are defects that the testers finds before a release to the customer, and those that the customer find after they have it in production.

There are 2 ways to try to reduce the number of bugs:

1. Fix each bug as it comes up. This works for most new bugs, but not the systemic issues that are throughout the system. One example is consistency issues in the GUI.

2. Refactor pieces of code to eradicate the bugs. There are not usually unit tests for legacy code, so the first thing you need to do is write them – according to what it should do, not what it does. The same with acceptance tests. Too often, the tests and documentation reflect the way it actually works and not the way it should. This also has to be addressed.

I mentioned the need for a defect tracking system earlier. One of the important uses of a tracking system is for generating client reports. Clients want to know which bugs were fixed and released in the latest version. Most clients want to test and to make certain the fixes were done to their satisfaction.

Documentation

If you are lucky, your legacy system has up-to-date user manuals: either a paper version or on-line help. If not, you have to try to fix the documentation along with the application and a good technical writer is absolutely necessary.

Existing Clients

Legacy systems usually mean existing clients. As part of the culture change, you need to convince customers that they should work with your new process. However, be aware that existing customers may have their own process for working with vendors and software releases. They may be used to getting status reports, design specifications, etc. You may need to adjust your processes to work with the clients.

> Currently we are working with a customer for whom we are doing some customized changes. Part of their requirements includes a change specification for all new features. Our Product Engineering group attempted to work with the customer and write the specifications prior to development starting a story. It worked for a little while until changes were made as development progressed. We then ran into the issue of testing to changing requirements, so we adapted our process. The customer still needs the specification for their process, but that didn't mean we needed to do it first. We talked with the customer, developed and tested the features, and only then, did the Product Engineering group create the specification.

Regression Testing

What about all the regression testing that needs to be done? The QA group has been responsible for ensuring that existing functionality has not been broken from one release to the next. Now they are expected to do this every two weeks at the end of an iteration. This is a tough problem to solve. If a team has automated regression tests, they are ahead of the game. If not, they need to plan for it.

> We chose to develop our own system test framework using Ruby since we needed to test a Java application as well as an embedded platform. The test team wanted to automate each story as it came up, but where the development team could concentrate on the stories, the release cycle and the legacy system claimed much of the test team's time. The test team coined the phrase "Regression Hell" for the seemingly never-ending test/fix/test cycle.

The problem we experienced is common when working with legacy systems. Testing without automation cannot guarantee that functionality didn't change. Legacy systems grow over time and usually are quite large, which makes testing for a release a nightmare. The problem is enhanced due to the number of bugs that exist in the system. They are not necessarily regressions, but may have lain dormant for many years.

A typical cycle can go something like this:

You've just released version 2.2.2 to the customer and are starting on the next release cycle.

Iteration 1: Stories are written, testers or customers create acceptance tests, pair testing happens, everyone is happy. The client reports a couple bugs, nothing serious – they can get fixed next release.

Iteration 2: More stories are written; the client starts more detailed testing, and uncovers bugs that need to be fixed before they can go to production with 2.2.2. The developers branch the code, and fix the bugs in both branches. The problem arises because there are no automated regression tests on the legacy system so the testers need to take a 2-week cycle to do regression testing on the 2.2.3 version with the fixes. This is where the problem starts.

What happens to the stories in this Iteration? They get behind because the acceptance tests are not written before the story is completed, let alone before end of iteration. Testing gets behind and stories are completed without testing.

Now the developers are in Iteration 3, and the testers haven't even caught up to Iteration 2. This is 'Regression Hell'. This is a problem that needs to be solved.

> We are attempting to solve it by taking developers to help with automating some of the tests for the legacy code. Instead of spending all their time working on new stories, the problem has been recognized by the team and has been given priority.

CONCLUSION

Software development teams that make a successful transition to an agile development process consider the organizational culture and the application being developed. They involve all the stakeholders, including those on the business and management side, and address their fears. They consider all the constraints; the size of the code base of any legacy application, the existing defect load, and documentation issues. They take into account any existing client base and plan how they will work with external clients. They tackle the tough issues around regression testing. Each team needs to find its own approach and make many tough decisions. By identifying potential problem areas up front, and continually reflecting on progress so they can tweak their process as they go, teams can successfully make this change.

Agile Facilitation

Esther Derby - Editor

Bringing Servant Leadership to Agile Software Development Projects - Jean Tabaka

BRINGING SERVANT LEADERSHIP TO AGILE SOFTWARE DEVELOPMENT PROJECTS

Creating the collaborative culture that supports agile teams doesn't happen by accident or in a vacuum. But agile software development team leaders can't nurture this vital collaboration environment by acting in traditional hierarchical ways. Creating and nurturing collaboration calls for a different kind of leadership: servant leadership.

Servant leadership isn't a new idea. Robert Greenleaf wrote of this new notion of leadership in the 1970s in a series of works that have influenced both Peter Senge and Steven Covey. We too can take advantage of Greenleaf's insight on how to serve our teams.

WHO IS THE SERVANT-LEADER?

Simply stated, the servant leader is always servant first, and a leader second. As it happens, the greatest service is often providing leadership. This reversal of priorities can prove challenging but ultimately rewarding in agile development projects.

Which are you - Leader-first or Servant-first? Here is a simple test:

> Do you make sure that other people's highest priority needs are being served first?

As you apply this test, think of the principles that have guided the Scrum Masters and XP Coaches you've observed.

CHARACTERISTICS OF THE SERVANT LEADER

What characteristics should you strive for in order to act as a servant in a leadership role whether in agile software development teams or leading entire IT organizations? Greenleaf posits 10 strategies you can learn (really!) in order to become a true servant leader:

1. Goal Setting

Always have a larger aim, a greater purpose to guide the smaller, tactical targets. A servant leader sets her sights on a greater sense of what the project is about, what the team can accomplishment, and therefore, how to guide the team day-to-day. A greater purpose allows the team to move forward without being bogged down in the ever-changing lay of the land. The goal sits above the release, iteration, or sprint. It is larger than the system metaphor; it is the metaphor for the team and its sense of self and reward. If you are unable to set an appropriate goal, none of the rest of the following strategies will matter.

2. Principle of Systemic Neglect

Servant leaders have a knack for highly focused prioritization of what needs to be done, accompanied by the equally important ability to "neglect" everything that is not useful to that priority. As items are managed and resolved, the previously neglected items move into the limelight and receive the full attention needed to solve them. Seasoned XP Coaches and Scrum Masters are very familiar with this notion of systemic neglect. It preserves their sanity as well as the sanity of their teams. It allows them to block out all "noise" that doesn't directly contribute to the success of the current sprint/iteration.

3. Listening

Ask about the expertise of a good coach or Scrum Master, and most likely you will hear that, in no small part, it is an ability to ask effective questions and really listen to the answers. Servant leaders hone this skill that is so antithetical to assertive, controlling leadership. Greenleaf tells us that in all his work with teaching management skills, teaching managers how to listen accomplished more than anything else he did.

4. Language as a Leadership Strategy

Servant leaders must be articulate, if for no other reason than to grasp the goal of the group and to evangelize it early and often within the team and beyond. Servant leaders need language to help team members reach consensus by helping them understand what might otherwise look like opposing views. Leaders use language to communicate face-to-face with team members, with stakeholders, with executive sponsors, and so on. The more skilled we are with language, the more effective these communications become and remain.

5. Values

Servant leaders have the attribute of being responsible, in that they are responsible for building, not destroying. They are responsible to their team, to the team goal, to the team's sense of self and its sense of accomplishment and reward. A servant leader's sense of values will naturally emanate to the team. As you develop your capacity for servant leadership, think through your values and live them.

6. Personal Growth

A servant as leader seeks growth experiences in how they can best serve the team both by continually educating themselves on their work as well as by tending to their personal needs. We see this in agile development teams as leaders read from diverse disciplines to better equipment them in technical settings. Servant leaders must be personally grounded before they can promote self-organization and self-empowerment of the team.

7. Withdrawal

A servant leader knows when to let go, when to retreat, when to trust the instincts of the team and to pull away. This balance ensures both that the team takes ownership of the work, and that the leader doesn't take on too much of the team's burden. Leaders who cannot rely on withdrawal as a strategy are not promoting collaboration and collective ownership of the team's success. At the very least, they perpetuate the role of hero. At worst, they disempower the team and revert to a command-and-control tactic in order to succeed at all costs.

8. Tolerance of Imperfection

Servant leaders learn to let go of their own sense of "perfection" or "right" and leave that definition to the team membership. The team resolves what must be done and what the "good enough" solutions are, not the leader. The leader simply guides them through questioning and listening to set the proper goal for accomplishment and then removes all obstacles in their path to that goal. This strategy is particularly vital in large IT groups where team make up is often pre-determined. Rather, the servant leader accumulates the talents and skills of the team and, given the team goal and the acceptance criteria, mentors team members to the defined deliverables.

9. Being Your Own Person

Ultimately, these strategies must always come back to the individual and how his personality works with project teams. This seems obvious but actually warrants emphasis. Each leader must make these strategies his own, define how they can work for him, and apply them in a genuine way.

10. Acceptance

The final strategy of the servant leader is acceptance. A servant leader has a duty to not only tolerate her own sense of imperfection, but also to encourage growth and motivation through unqualified acceptance of team members. Agile software development teams are so inter-reliant that they must have a fundamental sense of acceptance from the leadership down to each individual in order to truly encourage collaboration among the team members. Collaboration and acceptance go hand in hand.

PULLING IT ALL TOGETHER

As you consider your role in agile software development projects, reflect on these simple strategies from Robert Greenleaf. These strategies need not apply just to the leader. You can, as a true servant leader, groom team members as servant leaders by introducing them to these strategies over the course of the project. In fact, team members can support servant leadership as servant followers who groom their servant leaders by guiding them in the 10 strategies named here. They insist on leaders who actively engage in these strategies day-to-day with the project team.

Decide which of these strategies you can apply immediately in your leadership style. Revisit Robert

Greenleaf's strategies from time to time and check in to your true sense of service to your team as their servant leader.

MORE READING

You can learn more about servant leadership in these two excellent books by Robert Greenleaf:

"On Becoming a Servant Leader: The Private Writings of Robert K. Greenleaf" San Francisco: Jossey-Bass, 1996.
"Servant Leadership: A Journey into the Nature of Legitimate Power & Greatness" Mahwah, New Jersey: Paulist Press, 1977.

Agile Project Management

Kent McDonald - Editor

Communication and the Pragmatic Project Leader

Kent McDonald

INTRODUCTION

Software development is very much a team sport, which means that the ability of the team to communicate with each other and those that have interest in the project is a vital factor in the project's success. Because of this, perhaps the most important aspect of the Pragmatic Project Leader's role is to ensure that the project environment supports effective communication between everyone involved in the project. Agile Software Development methodologies put a great deal of focus on the teamwork and collaboration within the project team and therefore have several values and practices centered on fostering effective communication. In this article I take a look at why communication is so important to software development. Then I describe what I think effective communication means. Finally, I provide some tips gathered from my experience and the practices of Agile Software Development methodologies that may help you practice effective communications.

THE IMPORTANCE OF COMMUNICATION

Several of the major activities in software development require the team doing the development work to be good communicators in order to succeed. I'll touch on four such activities in this article that really point out the need for effective communication. These activities include determining requirements, building systems, resolving issues, and managing expectations about the project. I have no doubt that other activities in a software development project require communication, but these four are cases where the manner in which communication occurs can have a tremendous impact on the outcome of the project.

The impact of communications on determining requirements should be obvious to most people who have been involved in a software development project. After all, how will you know what to build unless you talk to the customer or end user about what they want or need? Recommend methods and practices for determining all have some element of direct communication with the customer or user, whether it is through interviews, surveys, focus groups, or Joint Application Development (JAD) sessions. Agile methods recognize that the communication surrounding requirements extends past the initial identification to include the need for clarification and understanding of the requirements as development is underway.

This is one of the purposes of XP's customer on site practice - to provide the project team with the opportunity to continue communicating with the customer about requirements on an ongoing basis, in the most effective means possible - face to face.

Whereas determining requirements requires a great deal of communication between the project team and the customers of the system, building systems adds on the need for the project team to communicate with each other, while still maintaining open communications with the customer. The people working together to build a system should be in constant collaboration, whether it is conferring over the design approach, helping others resolve design questions, discussing alternative coding issues and arriving at the best solution, or working together through pair programming to code various parts of the system. It is only through this constant collaboration that a group of people can effectively come together as a team.

When it comes to resolving issues, communication is the proverbial double-edged sword. When done properly, communication can help resolve issues, or prevent things from becoming issues in the first place. Just as often, however, poor communication is the cause of many issues in software development projects. The key to the impact of communication on issue resolution (or creation) is whether it was done at all, and whether it was done properly. Not communicating typically creates issues as Kent Beck points out, "Problems with projects can invariably be traced back to somebody not talking to somebody else about something important." (1). Communicating in the wrong way can also cause issues, or at least exacerbate existing ones, so people on the project team need to make sure that they are communicating properly and in the right frequency. I'll cover more about communicating properly below when I talk about effective communication.

Managing expectations can be the most difficult of all the communications related activities, because it depends a great deal on how well the project team communicated while doing the other activities. Even more so than resolving issues, the manner in which communication occurs can play a big factor in how successful your expectation management efforts are. If you are constantly communicating with your customers and stakeholders, but are viewed as being condescending to them, they will have a tendency to tune out what you are saying and form opinions about the result of the project based on their opinion of you. Personal experience has taught me that the best way to manage expectations with your customers is to practice upfront and honest communication with them. Telling the truth is always easier than fibbing, mainly because it is a lot easier to remember what you said.

EFFECTIVE COMMUNICATION

Effective communication is the practice of relaying relevant information to the correct people in a clear manner to avoid confusion and misunderstanding. The key elements of that definition are:

- Relevant information

Keep the message succinct and concise so that the important information is not lost in a lot of clutter. In other words, keep a high signal to noise ratio.

- Correct people

Make sure to send the message to the people that need to receive it. That means do not exclude people who should be receiving the information, but that also means do not relay information to people who really have no use for it. That is not license to keep information from people, but it is important to not flood people with too much irrelevant information.

- Clear manner to avoid confusion and misunderstanding

Pick the mode of communication that will ensure that information is easy for the receiver to understand

and process.

Face to face communication is the preferred mode of communication in most cases. This is especially true in the case when you have questions to ask or issues to resolve. This synchronous communication approach is the most effective means for solving issues because it is the best way to ensure that both the content and context of the message are being appropriately relayed.

The content of the message has obvious importance when it comes to avoiding misunderstandings. Face to face communication helps to ensure clear content because when two people are conversing and one person is not expressing an idea clearly, the other person can ask questions and talk through the issue until they reach clarification. In other communication methods, such as email, there is a lag in time to ask and answer questions, during which time misunderstandings can grow.

Context impacts the effectiveness of the communication because it makes it possible to convey the mood of the people communicating when words do not. Face to face communication allows the inclusion of emotions and tone whereas those subtleties are lost or misconstrued when the message is transmitted via bits and bytes. Often times the reader applies emotions to the message that the sender did not intend. This also can lead to misunderstandings, which in many cases constructs barriers that get in the way of what the message was really meant to be.

The framers of the Manifesto for Agile Software Development and the Project Management Institute agree with each other when it comes to favoring face to face communications. One of the principles of the Manifesto for Agile Software Development states: "The most efficient and effective method of conveying information to and within a development team is face-to-face communication."(2) The Guide to the Project Management Body of Knowledge, in the section about stakeholder management asserts: "Face to face meetings are the most effective means for communicating and resolving issues with stakeholders." (3) Jim Highsmith agrees that face to face communication is important, but also believes that some documentation is useful, "Understanding comes from the combination of documentation and interaction, or conversation - the conversations among people who have a certain knowledge." (4)

TIPS FOR EFFECTIVE COMMUNICATION

Listed below are several hints I have for practicing effective communication. You'll notice that several of them have to do with email. I included so many comments focusing on email because I believe that the reliance on email as a primary communications tool has caused more problems that it may have resolved. These are practices that I strive to follow in my everyday work; sometimes it is easier than others.

Never use email to communicate a time sensitive subject or issue. If a delay in resolving an issue will be detrimental to the project, talk face to face with the person who holds the key to solving the problem. Email is not a real-time communication mechanism; just because I sent an email today, does not guarantee that it will be acted on or even read today. The time lapse waiting for a response to the email can cause delays to the project. Issues that could be resolved in a simple five or ten minute conversation can take days or even weeks in back and forth email conversations that languish in electronic purgatory.

Do not use email if the information you want to convey is of a delicate or complicated nature. Face to face communication help resolve issues a lot quicker than conversing via email. There is less opportunity for misunderstandings, or stated another way, there is more opportunity to resolve misunderstandings without the lag time required for sending messages back and forth. Face to face conversations also allow for the use of white boards or scratch paper to collaboratively work on issues, sketching things out to help understanding.

Email was not intended for problem solving. Email is good for communicating precise information to a large group of people when the purpose is to record a decision or provide information to a group of people. It has been my experience that people typically do not read email very closely, so you want to make sure that the message is short and concise.

Involve only those who need to be involved in communications. This tip includes both face to face discussions as well as email. I have sat in on several status meetings that degenerated into problem solving sessions or theoretical arguments between a small subset of the people in the room. All this serves to do is waste the time of all the other participants. The "Parking Lot" for discussions comes in real handy in these situations. The parking lot is where someone suggests that the topic that is drawing out into Senate filibuster proportions be moved to the "parking lot" to be discussed at a later time by only the people who care. The equivalent in the electronic world is the practice of copying everybody and their brother in on email messages flying back and forth. This is usually done to cover someone's hind end but usually ends up creating mountains out of mole hills, and gathering "help" from people whose "help" is not needed or is often detrimental to the resolution of the issue.

You can lead a coworker to email, but you can't make them read. If you need to communicate the same message to several different people, use the advantages of email, just don't expect everyone to read or comprehend your message. If there are some people in the distribution list who absolutely must see and understand your message, follow up the email with a phone call.

The communications plan is a means to an end. The value in creating a communication plan as recommended in the PMBOK Guide is not in having the plan, it is taking time to think about whom you need to communicate with and what is the best method of communicating for them. If you are able to consider the items that the PMBOK Guide suggests at the time you need to do the communication, don't waste your time and effort creating the plan. Remember - the key to success is not how well you plan, but how well you execute and adapt.

Use the right means of communication for the situation. Know when to use face to face discussions, phone calls, or emails. Strive to use face to face communications whenever possible, especially when it is critical that the message be clearly understood. In cases where face to face communications is not practical, such as when the project team is distributed, try conversing via the telephone. At least then the communication is synchronous and not all context is lost in the conversation. Email still does have some good uses, especially when you need to convey information to a large number of people for information purposes, such as status reporting. Keep in mind when using email that the message should be written in a clear and concise manner.

The physical environment has a big impact on facilitating effective communication. Several agile software development methodologies stress the need to have the team co-located to encourage communications. Having everyone in the same room is not enough though. The area needs to facilitate communication by removing barriers that prevent team members from collaborating. Craig Larman discuss the concept of the "Common Project Room" in his book *Agile and Iterative Development*(4) that provides an excellent practice for facilitating communication through the right environment.

If you must have a record of a conversation, follow up a conversation with an email summary. That way you can make sure all parties involved are in agreement, and then the email serves as the record for people who have bad memories such as myself.

Face to face communication does not just have to be in a formerly scheduled meeting.

Some of the most powerful and most effective problem solving occurs in unplanned conversations in the hallway or one team member stopping by the desk of another to work something out. Encourage your team to not wait until a status meeting to bring up issues.

Don't waste the project team's time with written status reporting. If your organization requires project leads to send status reports to others in the organization outside of the project team, gather your status reports by talking to your team members instead of requiring them to send you updates. There should only be one status report created by a project team. Any more than that and time is being wasted. The daily Scrum meeting is an excellent example of this approach.

If all else fails; unplug the email servers for a couple of days. Force people to communicate using means other than email.

CONCLUSION

Effective communication is a cornerstone for effective projects. The ideas I provide above may seem rather straight forward and obvious, yet I am amazed at how much discipline it takes to actually follow my own advice on a daily basis. I humbly offer these tips up for all those who like me need a little reminder every once in a while what it takes to be an effective communicator.

REFERENCES

(1) Beck, K. *Extreme Programming Explained: Embrace Change.* Boston, MA: Addison Wesley, 2000.

(2) *The Manifesto for Agile Software Development.* www.agilemanifesto.org

(3) *Guide to the Project Management Body of Knowledge.* Project Management Institute, 2003.

(4) Highsmith, Jim, *Agile Software Development Ecosystems.* Boston, MA: Addison Wesley, 2002.

(5) Larman, Craig, *Agile and Iterative Development: A Manager's Guide.* Boston, MA: Addison Wesley, 2004.

About the Author

Kent J. McDonald works as Business Analyst and Project Leader consultant for Genesis 10 in Des Moines, Iowa USA. The views expressed are his own and are not necessarily those of his employer. You can reach Kent at kent@madsax.com or his website www.madsax.com.

Contrasting Scrum and DSDM's Approaches to Handling Mid-iteration Changes

Harprit S. Grewal
Department of Computer Science, The University of Calgary
grewal@cpsc.ucalgary.ca

Abstract

Requirements changes or "scope creep" has traditionally been attributed as the main reason for the failure of great many projects. The methods of the past have tried to tackle this issue in order to provide satisfaction to all stakeholders. None have made an impression as have the Agile methodologies. These methodologies or frameworks welcome changes in the middle of the project. In fact, changes are seen as an opportunity to showcase there ability to adapt. Scrum and DSDM are two of the core methodologies of the Agile alliance. In this paper we will discuss and contrast how Scrum and DSDM handle changes to requirements in mid-iterations.

Keywords: *Requirements, Agile, Scrum, DSDM, Iteration, Dynamic, Scrum*

1. Introduction

Software development is a complex activity. Even though software engineering is a relatively new area compared to other engineering fields, it has seen metamorphic changes in the way software is developed. Not too long ago, computers had ushered in an era of automation and promised to deliver the world to the businesses (and they delivered). However, software projects have seen their share of failures. These failures brought frustration to those who developed software and those who relied on the solutions.

As the saying goes "necessity is the mother of invention", efforts were made to search for the "holy grail" or the "silver bullet" of software development. High-level programming languages, object-oriented languages, CMM, and now Agile have (or had) shown a glimmer of hope. We are far ahead from where we were forty (or even twenty) years ago. So far we still do not have a silver bullet and perhaps we will never have one. What we do have is a better understanding of the dynamics of our field.

Software development has been defined as "empirical" (theoretical) rather than "defined" (black box) [1]. It is this dynamic nature of software development that has been attributed as a major cause of great software project failures. As Kent Beck says in his book on XP [2], "Change is the only constant…Everything in software changes. The requirements change. The design changes. The business changes. The technology changes. The team changes. The team members change. The problem isn't change, per se, because change is going to happen; problem, rather, is the inability to cope with change when it comes."

In this paper, we will discuss how two of the most popular Agile methodologies – Scrum and DSDM (calling it a framework is more appropriate), deal with requirements and how they handle changes to these requirement during an iteration.

2. DSDM

Dynamic System Development Methodology or DSDM is a high-level framework for control of processes rather than a technique. Its aim, just like other agile development techniques, it to deliver value to the customer. In doing so, it strives to satisfy needs of all the stakeholders, be it user, management, developers, or customers. DSDM achieves this by using different techniques available to the framework and by "flexing requirements". It considers delivering a working product to the customer early on than wasting time and resources on trying to address all the possible situations that the project may encounter.

The basic principles of DSDM are described below:

- Active user involvement is important
- The team must be empowered to make decisions
- Frequent delivery of usable products is important
- Fitness for business purpose is the essential criterion of acceptance of deliverables
- Iterative and incremental development is important
- **Changes in development are reversible**
- Requirements are baselined at high level
- Testing is integrated throughout the life-cycle
- Collaboration and cooperation between stakeholders is important

DSDM is based on the fundamental assumption that useful and usable 80 percent of the functionality required for a project can be completed in 20 percent of the time estimated to complete the project. DSDM recognizes the fact the requirements at the start are inaccurate and trying to satisfy all the requirements may result in the project getting delayed and/or over-budget and may fail to deliver business value.

3. Flexing Requirements in DSDM

DSDM asserts that requirements that will be satisfied can change as the business needs them to. As complete turn around from the traditional approaches, DSDM make the time factor a constant. The resources are assumed to be constant to a certain extend. The only aspect that is allowed to vary is requirements because, as stated above, requirements can be inaccurate or incomplete and a product developed based on these inaccurate requirements is flawed as it does not deliver business value.

4. How DSDM manages requirements

The techniques used in DSDM fall into two broad categories. First consists of the project techniques that would apply to all projects regardless of the nature of the project. These techniques are generally concerned with managing the requirements. These include Timeboxing, Priority Requirements Lists (PRL) and the MoSCoW. Other techniques are concerned with micro-managing the project and would depend on the individual nature of the project. Since this paper is concerned with how DSDM manages requirements (at mid-iteration), brief mentions of the former techniques are in order.

4.1. DSDM MoSCoW technique for requirements management

MoSCoW technique is used to prioritize the requirements where M stands for requirements that are Must haves (requirements that must be satisfied in order to deliver value); S stands for Should haves or requirements that are important but do not hold back the development of the project; C is for Could haves or requirements that would be nice have; and W is for Won't have or requirements that would not be addressed in the current interation. During the phase of requirements prioritization, 60 percent of effort is expended on recognizing the Must have which forms the minimal usable set of requirements.

4.2 DSDM Timeboxing

Timeboxing is an important concept in DSDM. The project timebox is defined as the time between the start of the project to the end date of the project. The end date is a fixed date and the date by which the working system or part of the system must be delivered. DSDM nests the concept of timebox in an iteration. The goal is to provide a series of fixed deadlines by which part of the system must be delivered. It is important to note that the focus is not on activity but on actual production of a part of the software that brings value to the customer.

4.3 DSDM Priority Requirements Lists

The Prioritized Requirements (PRL) List defines what the proposed solution must do and how well it must do it. It provides the foundations for all planning decisions throughout the project [3]. The PRL defines the scope of various iterations by focusing on the requirements and the business processes that need to be supported. This document (PRL) is a common source for business and development team for referring to the priorities. Thus, it is referred throughout the project and continuously updated.

PRL, among other things, differentiates between the core and additional functionality, a minimal usable set of requirements, the alternatives to Must haves. PRL uses MoSCoW prioritization as its basis. Since PRL consists of a mix of Must haves, Should haves, and Could haves, it gives the development team some room to replace some Should have and Could have requirements with some Must haves if these happen to creep in during an iteration. The replaced requirements become part of the future iterations depending upon their priorities which in turn may displace some other requirements in those future releases.

5. Everything's a Must Have!

In a document named "Everything's a Must Have!' [4], Kevin Barron and Mark Simmonds have outlined the results of a workshop aimed at common problems around determining the requirements and delivering the requirements. This document states the reasons for common problems associated with requirements and then suggests solutions on how these should be addressed. An important aspect of the outcome of the workshop, as it relates to this paper, is that businesses consider almost all the requirements as "Must haves". The reasons for behavior are manifold. First, businesses are not aware of the definition of a "Must have" requirement. Second, businesses insist on getting all the requirements delivered mentioned in the contract. Third, similar to the last point, that the detailed specification identifies each item as a Must. Fourth, the requirements have been prioritized at very high level that all of these requirements are considered Must haves. Fifth, the sales team oversells the ability to deliver solutions. Finally, the users find it difficult to see beyond there immediate needs and insist on getting every associated with their area done.

The common solution to all these causes is education of all parties. The business users should be educated about the costs of making all the requirements as must have, the sales people should be educated about the development approach and how overselling can hurt the reputation of the organization. Benefits of changing of what was originally agreed upon to deliver business value should be sold to the users.

These reasons have important implications for mid-iteration changes. If these could be addressed early on, then the users will be better informed to reduce their share of mid-iteration requirements changes (if they are not of type Must haves). Thus DSDM takes a pro-active approach to reduce mid-iterations in the first place.

6. Scrum

Scrum is one of the most popular Agile practices. It is iterative and incremental process that can be applied to any project be it software or otherwise.

The important terms when talking about Scrum are succinctly defined below:

- The Daily Scrum – Meetings held every day during the month long sprints where team members discuss the work they did the previous day, the work they intend to do at the day of the meeting, and any obstacles that may hinder their progress.
- Product Backlog – A list of all functionality desired in the product.
- Sprint Backlog – A subset of Product Backlog which form the tasks of current iteration. Chosen by the Scrum team based on highest priorities and time constraints.
- Product Owner – Person from marketing or a user. Prioritizes product backlog.
- Scrum Team – Members of a team responsible for completing tasks in a sprint backlog. There are no special designations within the Scrum team and all team members have one goal – achieving the sprint's goal.

Scrum's main goal is to provide a shippable (usable) product at the end of each iteration.
Scrum achieves this by prioritizing the requirements in a *Product backlog* which is further broken down into *Sprint backlogs.*

Each *Sprint* is an iteration which is ideally 30 days long. During each Sprint, the development teams meets daily for about 15 minutes to discuss what each team member accomplished the previous day and what he/she planned to achieve that day. Also, they should discuss what problems they encountered during their work. It is the duty of the Scrum Master (project manager) to remove these obstacles so that the team can focus on what they are supposed to do – deliver value. Scrum can be used as a wrapper around other processes such as the development focused eXtreme Programming to help achieve better quality code and individual training.

7. Requirements Handling in Scrum

Scrum is about empowering teams. At the start of a Sprint, a Sprint Planning Meeting is held in which the development team slices off the highest priority items and make them part of the Sprint Backlog. The team may, however, pick some low priority tasks that are directly related to the high-priority tasks. This may seem similar to picking up the Must have and Should and Could have from the MoSCoW prioritization of DSDM.

Since, by definition, the development team selects the items that are of highest priority to the Product Owner and commits to finish these tasks before the end of the Sprint. This commitment is in return for a commitment from the owner that he/she will not throw in new requirements during the Sprint. The team welcomes and encourages changes as long as they outside the scope of the current Sprint. Once the team starts on a Sprint it remains maniacally focused on the goal of that Sprint. Once the team starts on sprint it remains maniacally focused on the goal of that Sprint [5].

The rationale behind this argument can be understood from a case study which be found in the Scrum website [6]. This case study discusses a project that was late and had too many bugs in the beta release. Instead of the bug/issue list getting shorter it seemed, as time passed, the list remained practically the same. The reason, as it was discovered later, was that since the project was late, marketing and product manager were introducing functional enhancements which sidetracked the developers from attending to the real issues. The issues were entered into the bug/incident tracking system by all and high priorities were assigned to them. As a first change to the operations, the priority field in the bug tracking system was made secure so that only one person would be able to enter the issues/bugs based on their criticality in stopping the delivery of the system. The policy of only concentrating on issues that would help the product become more stable was enforced. This helped in reducing the number of high-priority issues and the product was eventually released to the customer.

An argument supporting the above observation is that change requested will have to be weighed in comparison to the progress being made in the current Sprint. If the change is extremely important, the current Sprint can be abnormally terminated. The customer should, however, be presented with the choices and the consequences of these choices should be laid out.

Recently, I posted a message on the Scrum mailing list asking what people's take was on how Scrum treads the line between keeping the customer happy by incorporating changes and at the same time keeping the promise of delivering the functionality at a fixed data as promised at the start of the Sprint. In his reply, Jeff Sutherland mentioned the use of "Dynamic scrum" whereby the backlog is managed in real time and the consequences of delivery date getting late are known to all instantaneously. The Product Steering Committee is responsible for making changes to the backlog and developers focus only on the items that are part of the backlog. "Dynamic Scrum requires total automation of backlog so current state can be viewed in real time in a single system that everyone uses. Change impact can be seen in real time. This is similar to Cisco's claim to be able to balance their books in real time at any time. It is not recommended without this high level of automation." [7]

In a separate message, Mike Cohn suggested that the Product Owner should be asked two sprints ahead if there could be more changes needed from what has already been decided and date is given to the Product Owner to think about it. This, Mr. Cohn suggests, "helps avoid abnormally terminating a sprint (or having bad feelings about what did/didn't get done in a sprint." [8]

8. Contrasting DSDM and Scrum for Mid-Iteration Requirements Changes

As we have seen in the discussion about requirements handling of the two methodologies, DSDM is more flexible in terms of incorporating changes. Based upon MoSCoW prioritization, where 60 percent of the requirements are Must have and the remaining are Should haves and Could haves, the team can replace the non-showstoppers with the most pressing requirements if they must be included in the current iteration. Of course, this must be with the assumption that the person defining the priority of the requirement to be incorporated knows the definition of a Must have requirement (see section "Everything's a Must Have" above). If the users are just concerned about their narrow area of work and do not see the overall benefits of the project and categorize everything as high priority, then it will be very easy to loose sight of the goals for the project. However, there is an aspect of planning required on the part of the development team to be able to accommodate the requested change. There is no mention in the literature researched about the cases when there is a deadlock sort of situation where the new requirement is highly critical but trying to accommodate it in the iteration would surely result in a delayed iteration, something that is a taboo in DSDM.

Scrum, in contrast to DSDM, uses the highest priority items from the Product Backlog to form the sprint backlog. Once the sprint backlog is decided upon, the team "insulates" itself from changes for the duration of the current Sprint. The changes requested in the mid-iteration form part of the next sprint backlog. This approach has it pros and cons. A pro being that the team is not distracted from dealing with high-priority issues with something like "functional enhancements". A con, on the other hand, could be that a high-priority may have to wait until the Sprint is over. However, as mentioned above, the choice must be left to the product owner whether to abnormally terminate the current Sprint or wait for a couple of weeks for the Sprint to get over and then re-evaluate the Product Backlog.

9. Conclusion

Agile methodologies such as DSDM and Scrum are built around the best practices that have been around for quite some time. These practices have been formalized in these light-weight Agile methodologies. The proponents of these methodologies are not selling them as silver bullets, but only as guidelines with the assumption that the implementers will customize according to their needs. Requirements will invariably change as the needs of the businesses change. If the business could get what they wanted when they wanted, there won't be any change requests. I do not see how this can be done (or else we won't be discussing all this). Unfortunately, there is a time lapse between when a project is initiated and when it is delivered. What we can try to do is be flexible in the way we develop software so that we not only welcome change, but encourage it.

Scrum and DSDM offer very contrasting ways of handling requirements changes and each has its merits. Therefore, it is hard to tell which is better. It really depends on the situation and the customer you are dealing with. What is common in these methodologies, and in fact amongst Agile methodologies, is importance of user involvement and customer education. If the customer is educated, he/she would not make unrealistic demands. If they still insist on demands that are unrealistic, then they should be made aware of consequences their decisions may have. It is our duty to educate our users and customers. If the customer is better informed of what is involved, chances are that they will cooperate in the development effort. After all, who does want to be part of a successful project?

10. References

[1] Development: Empirical or Planned?
http://www.controlchaos.com/debate.htm

[2] K. Beck, eXtreme Programming eXplained, Addison-Wesley, Reading, Massachusetts, 2000

[3] Developing a Prioritized Requirements List (PRL) http://www.dsdm.org/timebox/issue9/prl.asp

[4] K. Barron & M. Simmonds, *Everything's a Must Have.*
http://www.dsdm.org/timebox/newsletter7/musthaves.asp

[5] The Scrum Development Process
http://www.mountaingoatsoftware.com/scrum/

[6] Scrum Principle: *Backlog work can come from many sources, but only one person prioritizes it. Case Study 1*. http://www.controlchaos.com

[7] Scrum mailing list message
http://groups.yahoo.com/group/scrumdevelopment/message/2888

[8] Scrum mailing list message,
http://groups.yahoo.com/group/scrumdevelopment/message/2895

Dealing with resistance to the DSDM approach

Kevin Barron Independent Consultant and Neil Bennett Xansa

Resistance is futile! unfortunately not always true when it comes to introducing DSDM into an organisation. This stage brings with it many challenges. In fact there is an entire white paper from the DSDM Consortium dedicated to this issue. The issues and solutions documented below came out of a workshop we ran at a DSDM Roadshow and represent the experiences of many users of DSDM. Of course there will be times there is no way round the resistance and you will concede defeat if you want to keep the client. But hopefully some of the solutions offered below will help you overcome resistance in many cases.

ISSUE	POSSIBLE SOLUTION(S)
Lack of customer commitment at ground level.	Use Impartial facilitation. Schedule some facilitated workshops to get everyone involved and get buy in from all stakeholders.
Resistance to changes in requirements.	
A Timebox? That's just another deadline!!	Implement the idea of personal time boxing this may carry over into the project.
Timeboxed to death.	Time boxing requires dedication people may need reminding of this.
Unrealistic time boxing and lack of necessary prioritization.	Provide some examples of best practice to show the real business benefits of timeboxing.
I've been doing this job for 30 years and I've never planned my time yet!	
Great in theory but…	Don't sell the methodology sell the logic. Run some overview sessions and where possible practical sessions. Provide formal/informal training. Get some user groups started. Make sure you are educating the right people.
Things are hard enough without introducing a methodology.	
Inability to prioritize.	Focus on the business needs when you are prioritizing.
We don't have the skill/training or budget.	Provide support via DSDM clinics (health checks).
We'll have to send everyone on another Course and frankly we don't have the time or the budget.	Use a focused rollout and organic growth. Get some Key projects or Internal Projects then once these are successful build on these. Focus on business benefit achieved in your chosen projects
Change fatigue.	Rollout by stealth. Don't over publicize in this case. Do the softly, softly approach until you have positive buy in.
Project team members not adopting the roles.	Pick the right people for the early projects with the right skills. Mix DSDM and non-DSDM resources & encourage knowledge transfer.
Organisation makes a token gesture to do DSDM.	Get tough. Be clear in business terms, on risks/benefits). Raise issues. Don't understate priority. Be clear on requirements. Be prepared to de-commit.

Upcoming DSDM Events

Event: London DSDM Roadshow
Dates: 20 May, 2004
Type of event: Roadshow
Location: Hilton Paddington, London, UK
Organized by: DSDM Consortium
Contact: Linda@dsdm.org
More information: http://www.dsdm.org/en/about/agm2004.asp

AGILE MANIFESTO VALUES:

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

While there is value in the items on the right, we value the items on the left more.

Agile Alliance

www.agilealliance.org