
METHODS & TOOLS

Practical knowledge for the software developer, tester and project manager

ISSN 1661-402X

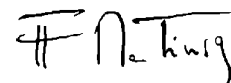
Spring 2008 (Volume 16 - number 1)

www.methodsandtools.com

Can We Develop Agile Software in Traditional Organizations?

As it was confirmed by our last survey, the adoption of agile approaches has been increasing recently. Following these results, I asked software practitioners on different discussion forums to share their opinion about the substance of agile adoption. As agility is now becoming "trendy", we could see a number of organizations that will qualify now themselves as "agile", without implementing the essence of the agile software development practices. The fact that everyone could have its own definition of "Agile" is not upsetting. After all, agile groups different approaches like extreme programming, scrum or test driven development. People could adopt the "agile" badge as they could have been doing "RUP" before, just because it is the "right" answer. Besides this, the main obstacle to true agile adoption is the organizational context existing in large companies or governments agencies. As a participant cleverly says, it is difficult to transition from a "Command & Control to a Collaborate & Communicate structure".

Drawing from an engineering perspective, many organizations wants to specify a solution and then estimate the time and budget needed to realize it. I suppose that you would not commit to build a new house without a detailed plan and a budget. Many managers don't act differently when they have to invest in a new software system. The agile approach requires relinquishing the (illusion of) long-term control to accommodate changes during the project. We have a long history of organizations signing off projects knowing that the probability of getting the initial requirements for the budgeted price is very low, and this is also true for other domains than software development. People don't ignore this situation, but the buyer mainly uses the project "contract" as a protection mechanism against criticism. On the other side, many sellers use the proposal only to get the contract signed, hopping to bargain about price and functions after the project has begun. We are sadly often more dealing with organizational politic than honest human relationships. This has caused often a situation of distrust between users and developers and transition open collaborative development is difficult. If things started to go wrong, managers don't want to find themselves explaining to their boss that they started a project with a lot of (assumed) uncertainty. The final question is: should we adapt organization to approach or adapt approach to organization and people? I support the agile trend to change the user-developer relationships, but I doubt that this will be a quick and easy goal to achieve, moreover in large organizations where the political aspect is more important. We should therefore keep in our toolbox different approaches to suit diverse organizational contexts. This is also why this issue will offer different visions to improve your software development process.



Inside

How Quality is Assured by Evolutionary Methods	page 2
OpenUP –The Best of Two Worlds	page 21
Real Reuse for Requirements	page 33
Creating an Agile Environment	page 41

Software Development Jobs

Methods & Tools introduces a new job board

<http://www.softdevjobs.com/>

Are you looking for an efficient, simple and cost effective tool to recruit experienced developers, Web designers, testers, database administrators or software project managers?

Your job post will reach an audience of 40'000 monthly Web site visitors and be mentioned in the future Methods & Tools issues that are delivered to a global community of more than 50'000 software development professionals.

Facts: Asia, North America and Europe host each around 30% of our registered readers. They work as developers (37%), project managers (27%) and in quality assurance (20%).

**Methods & Tools readers will
get a 50% discount with
discount code "MT01".**

Job seeker? Get automatic alerts of new jobs in your country or speciality via e-mail or RSS

How Quality is Assured by Evolutionary Methods

Niels Malotiaux, niels @ malotiaux.nl
<http://www.malotiaux.nl/nrm/English/>

1 Introduction

After several years of experience as a Project Coach introducing Evolutionary Project Management Methods (Evo) in development projects, I think I can claim that Quality can be Assured if projects apply these methods. Does this mean that the Quality Assurance function is not needed any more? No. QA is still needed, because one of the main factors jeopardizing the Assured Quality is lack of discipline - discipline to keep applying the methods in order to meet our commitments.

In many cases, people know the best way to do their work. However, if nobody is watching, people tend to take shortcuts. If somebody is watching over their shoulder, people tend to take fewer shortcuts. The Project Manager can watch over the shoulders of the team. The team can watch over each other's shoulders. But who's watching over the Project Managers' shoulder? This task is the responsibility of management, but the Quality Assurance function can help.

Even with an assurance function in place, team members still have to know what is the best way to do their work in the first place. Since there is no absolute "best way", while the "best way" is even dynamically changing, we must also provide the people with an ability to actively find out the best way while working in the project.

Evo is actually rapidly and frequently applying the Plan-Do-Check-Act (or Deming) cycle, not just for the development of the *product*, but at the same time for the organization of the *project* and even for assessing and improving the *methods* used on the project. We need to continuously ask ourselves: "What should we do *now*, in which *order*, to which *level of detail for now*".

Working the Evo way means organizing the work in weekly (or even shorter) Task-cycles. In these Task-cycles we optimize estimation, planning, and tracking. Task-cycles feed bi-weekly (or shorter) Delivery-cycles by which we optimize the requirements and our assumptions. We use a practice known as TimeLine to create and maintain the total project scope and to connect the Project Result, through the Deliveries, with the actual work organized in Tasks. Evo combines Estimation, Planning, Tracking, Requirements Engineering, Requirements Management, and Risk Management into *Result Management*. *Result* is defined as the combined value we provide to all the Stakeholders of our product, ultimately leading to customer success. Evo has a fanatical view on ROI: Whatever we do should contribute to the Result and we try to avoid whatever does not contribute

In this paper I will explain the basics of this Evolutionary approach and practical details people can start applying immediately.

2 The Goal

Let's assume that the purpose of development projects is to deliver *what* the customer needs, at the *time* he needs it, to create substantially *greater value* than the *cost of development* and to enable *customer success*. In short, we call this Quality On Time: the right things at the right time.


It is important to note that the *functionality* we are working on in most development projects *already exists*. Usually, all we are supposed to do is enhance the performance of specified functionality to create more value for the customer. The set of functions we are enhancing defines the scope of the project. The scope should be chosen such that it provides more value for cost than another scope.

Banks have banked for thousands of years. First using clay tablets, then using card-trays and now using computers. Banks are, however, still doing what they did before. The function is still the same, while the performance (ease, speed, complexity of transactions) is enhanced. If a new system does not deliver sufficiently more value than the old system, there will be no funds to pay for the new system and the developers.

It would be nice if we could in one project develop the ultimate solution, creating the ultimate value. Apart from the risk that, when done, we could be out of work, this is not possible because of limited resources such as:

- The available time (time to market may strongly influence time to profit)
- The available money
- The available people and the capabilities of these people (it would be nice if we could hire the best people. Normally, however, the challenge is to succeed with *average* people)
- The available experience on the subject
- The available technology
- The capability of the users to adopt the new system


Advertisement – Visual Use Case - Click on ad to reach advertiser web site



The graphic shows three overlapping, semi-transparent stars in blue, orange, and green on the left. A large, light blue arrow points from these stars to the right, where there is a solid orange circle, a solid green square, and a solid blue triangle.

Crystallize your Requirements

Document Requirements precisely with "Structured Pseudo-Code"
...and then with a single click...
...convert it into a "Flow Chart" that everyone can understand!



Visual Use CaseTM 2006
Write better Use Cases in less time. Multiply Productivity, Improve Software.

Click here to [download your instant free trial](http://www.TechnoSolutions.com) of Visual Use Case.
<http://www.TechnoSolutions.com>

In development projects we can only strive to optimize the compromise between value creation and the available limited resources. If the results we can achieve, given these limited resources, are insufficient to provide significant value for customer success, we shouldn't even start the project. Given these limited resources we are not even satisfied with *good* results, we actively want to *maximize* the Result created. Looking back at the end of a project, not only should our customer have a big smile of satisfaction, we should ourselves also be confident that we couldn't have done better.

This implies that we should feel a Sense of Urgency to constantly optimize the results we are working on, to constantly optimize our success. Without this Sense of Urgency, Evo doesn't work.

3 Plan-Do-Check-Act

Since childhood we learn intuitively through experience. Besides learning from our own experience, we also learn from accepting the experience of others: at school, in workshops and at conferences. This learning process is rather slow. We can, however, stimulate the learning process by actively using the Plan-Do-Check-Act cycle, as presented by Deming:

- **Plan** What are we supposed to accomplish and how are we going to accomplish it?
- **Do** Carry out the Plan
- **Check** Is the result and the way we achieved the result according to the Plan?
- **Act**
If the result was not according to the Plan, what are we going to differently the next time to achieve a better result?
If the result was according to the Plan, was it accidental? How do we make sure next time the result is equally according to Plan?

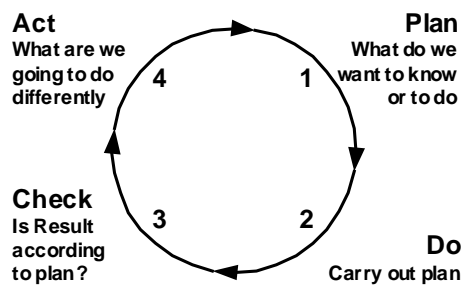


Figure 1: PDCA or Deming cycle

Do is never a problem: we “do” all the time. *Plan* we do more or less, usually less. For *Check* and *Act*, however, we have no time because we think we want to go to the next *Do*. Well, that's what I believed until recently. Taking a closer look at what really is happening we can see that *Check* is often done: people seem to be quite aware what is going wrong and often even know what should be done about it. The real problem is that we don't *Act*: taking what we know and doing something about it.

Sometimes I hear people in a project week after week complaining about the same problem, usually that somebody else is doing something wrong. My advice: either deal with it or stop complaining. Don't keep wasting energy complaining about the same problems over and over. Do something: Act! Find a solution, plan the time needed and solve the problem.

4. Evo

4.1 Evo

Evo is short for Evolutionary Development, Evolutionary Delivery, Evolutionary Project-Management, deliberately going through the Plan-Do-Check-Act learning cycle rapidly and frequently, for product, project and process, continuously thinking “*what* to do, in which *order*, to which level of *detail* for now”. It’s a label for a set of methods that allow us to effectively and efficiently run projects, delivering Quality On Time. Evo integrates Planning, Requirements and Risk Management into *Result Management*. It’s *actively induced* evolution because we don’t wait for evolution to happen, we make it happen.

Many organizations mandate a Project Evaluation at the end of every project. Even so, few projects do the actual evaluation because they feel that these evaluations do not contribute to better results. Why is this? Consider one-year projects (see figure 2). People have to evaluate what went wrong and what went accidentally right (and why) as long as a year ago. In addition, they may not be able to use the learning from an event until as long as a year after the fact. The idea of evaluation is valuable. The time constants of this process as described above are, however, beyond the capabilities of the human mind. In Evo, we do evaluations (PDCA) every week. This tunes the time dimension to the human mind’s abilities and enables us to rapidly implement what we learn.

Advertisement – Manage your Projects from End to End - Click on ad to reach advertiser web site

I AM A PROJECT MANAGER.
I MANAGE A TEAM ACROSS FOUR SITES.
I NEED MKS.



My team needs real-time collaboration across platforms and locations.

We need clear traceability throughout the project lifecycle, starting with requirements.

I need real-time updates on the status of my projects. I want project metrics in a dashboard view.

We need ONE application lifecycle management platform to manage our projects from end to end. We need MKS.

With MKS, we are one.

New in MKS Integrity:

- ✓ Test Management
- ✓ Requirements Change Management and Requirements Reuse
- ✓ SAP and PeopleSoft Change and Release Management

Call us: 519 884 2251 or toll free 1 800 613 7535 | www.mks.com

MKS

APPLICATION LIFECYCLE MANAGEMENT
FOR THE ENTERPRISE

4.2 Evo and the Product

We don't know the real requirements. *They* don't know the real requirements either. So, stop pretending we know, and accept that we have to find out what the real requirements are, together. This includes finding out who *they* are. We can make the nicest systems, given unlimited time and money. However, our customer doesn't have unlimited time and money. If the customer cannot afford all what is possible, we must find out the best Result we can achieve within the limited resources. If that's less than the customer needs for success, we shouldn't even start.

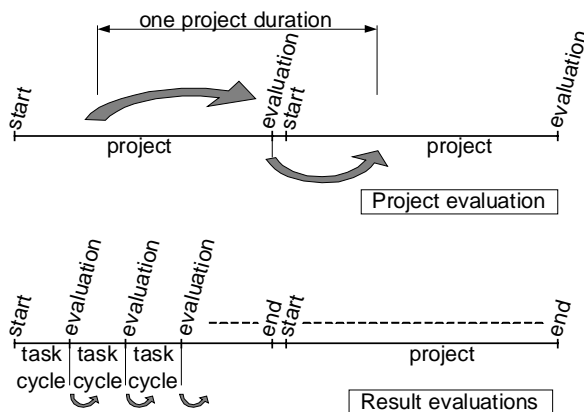


Figure 2: Project and Result evaluations

Result is the value gained by the use of what we developed. Result ultimately is customer success. If no value is gained, there is nothing to pay our salaries from. Because not all customers are aware of this, we have to work with the customer to find out what the optimum Result is to make sure that we are generating significant value. In Evo we work with a no-cure no-pay attitude. Whatever does *not* contribute to customer success, we don't do.

4.3 Evo and the Project

The optimum Result is the best product for the least cost. At the start of the project we don't know what the optimum Result is, so we must organize the project in such a way that we discover and implement the optimum Result at the lowest cost. This implies optimizing the effectiveness and efficiency of discovery and implementation. It also means that we have to change our estimation practice from optimistic to realistic, so that we can predict the future more accurately. We have to accept the realistic estimates and plan accordingly. We have to dynamically keep our plans up to date in order to keep control over the Result. We must learn better time management and better priority management. These are among many issues we can improve. In Evo we are constantly, dynamically improving on these issues because our success is at stake. We do not only design the product, we also *design the project*.

We take time and money budgets very seriously. This means that we don't ask for more when we were supposed to deliver. If the budgets really were insufficient we could have predicted this way before the budgets ran out and, together with the customer, we could have acted accordingly.

4.4 Evo and the Process

Because it is continually being improved as a process, Evo is made up of the best set of methods we know at a given time. If we find a better way, we change to the better way. Not only do we

employ PDCA on the product and project activities, we also constantly and dynamically apply the PDCA cycle to the methods we use. If another method seems better, we try it. We may experiment. But we deliberately Check and Act: if the new method is better, we change to the better method. If the new method is not better, we revert to the last known best method. Methods, processes and procedures are there to help us. If they don't, we discard them. A side effect is that Evo processes may be different between projects and between organizations because of differences in culture or differences in experience. The common property is always the urge for success in defined goals.

4.5 Does Evo cost more time?

Some people fear that all these evaluations, intensive planning and constant improvements will cost a lot of extra time. It does not: experience based on many projects proves that it *saves* time. Why else would we do it? The “extra” things we do in Evo projects are the things that should be done anyway on any project to make it successful. So, we don't really do “extra” things. We only do those things that contribute to Quality On Time.

4.6 When do you *not* need Evo?

There are circumstances where you may consider not using Evo, such as:

- The requirements are completely clear and nothing will change. *This is production, not development.*
- The requirements can be easily met with the available resources in the available time. *Still, Evo can make you achieve better results in shorter time.*
- The customer can wait until you are ready. *Still, Evo can make you achieve better results in shorter time. Why waste your time while you can do more interesting things?*
- The customer doesn't care about the result. *Should we contemplate this project? Is he going to pay?*
- You don't care about the cost or time. *Could be a hobby or a vacation.*
- Your boss doesn't care about the cost or time. *He probably doesn't know what to do with his money.*
- Management doesn't know what to do with the time saved. *Be careful, they may frustrate your project.*
- There is no Sense of Urgency.

Sense of Urgency is an important issue to watch for. Most people, including management, will immediately affirm the urgency of the best Result at the lowest cost. That's trivial. However, there are cases where their actions tell a different story. The remedy is either to educate them by coaching, or not to bother them with Evo. There are plenty of places where you can be successful with Evo, so why bother if they don't want to be more successful. Besides, Evo is never a goal in itself. Result is all that counts.

If they get optimum results their way, you shouldn't complain, but rather learn from how they do it.

JAZOON'08 INTERNATIONAL CONFERENCE ON JAVA TECHNOLOGY

**JUNE 23 – 26, 2008
SIHL CITY · ZÜRICH · SWITZERLAND**



Top Speakers:

Dion Almaer
(Google Gears), Google

Joshua Bloch
(Java Puzzlers), Google

Ludovic Champenois
(Open DS), Sun Microsystems

Roy T. Fielding
(REST), Day Software

Neal Gafter
(Google Calendar), Google

Rod Johnson
(Spring), SpringSource

Ted Neward
(Java and .NET)

Martin Odersky
(Scala), ETH Lausanne

Simon Phipps
(Open Source), Sun Microsystems

James Ward
(Flex), Adobe Systems

...and many more

Conference Themes:

Enterprise application technology

Web services

Glassfish

Mobile computing

Android

Web 2.0 technologies

AJAX, DHTML and mash-ups

Rich UI technologies

Testing

Virtualization

Security

Google Gears

More information and registration: <http://jazon.com>

JAZOON08

THE INTERNATIONAL CONFERENCE ON JAVA TECHNOLOGY
JUNE 23 - 26, 2008 ZÜRICH

Sponsors



canoo

AdNOVUM

Google

ORACLE

Partners

netcetera

eveni

JUG S



5 Evo basics

We organize Evo projects on several levels. We use the TaskCycle to organize the work, the DeliveryCycle to organize the Results and TimeLine for making sure we'll be on time.

5.1 TaskCycles

In the TaskCycle we organize the work. We are checking whether we are doing the right *things*, in the right *order*, to the right *level of detail*. We are optimizing our estimation, planning and tracking abilities to better predict the future. We select the highest priority Tasks, never do lower priority Tasks and never do undefined Tasks. As a practical rule, we plan 2/3 of the available time and in the remaining 1/3 of the time we do all those things we also have to do in the project, like small interrupts, helping each other, project meetings and many other things. If we plan 100% of our available time, we will still do all those other things, and we will never succeed in what we planned.

TaskCycles take at most one week, in some cases even less. Every Cycle we decide what is most important to do, how much time it takes to do it completely (we define what completely means) and then what we can do in the available time. We also decide what we will *not* do in this Cycle, because there is no time to do it. Now we can focus all our energy on what we *can* do, making us more relaxed and more productive. Some managers fear that planning only 2/3 of the available time makes people do too little. In practice we see people do more.

5.2 Task Selection Criteria

The following set of Task Selection Criteria proved useful for deciding the priority of Tasks:

- Most important requirements
- Highest risks
- Most educational or supporting things
- Active Synchronization with others outside your project

Remember: Every Cycle delivers a useful, completed Result.

5.3 DeliveryCycles

In the DeliveryCycle we organize Results to be delivered to selected Stakeholders. We are checking whether we are *delivering* the right things, in the right order, to the right level of detail. We are optimizing the requirements and checking our assumptions.

A DeliveryCycle normally takes not more than two weeks. Novice Evo practitioners, almost without exception, have trouble with the short DeliveryCycle. They think it cannot be done. In practice we see that, without exception, it *always* can be done. It just takes practice. One of the important reasons for the short length of the cycle is that we want to check our (and their) assumptions before we have done a lot of work that later may prove unnecessary, losing valuable time. Short DeliveryCycles help us do this with minimum risk and cost.

A common misconception of Deliveries is that people think they always have to deliver to users or customers. On the contrary, we can deliver to any Stakeholder: the user or customer, ourselves or any Stakeholder in between. This makes it easier to define Deliveries. However, we must always optimize Deliveries for optimum feedback: we must check what we are doing right and what we are still doing wrong.

Delivery Selection Criteria

The following set of Delivery Selection Criteria proved useful for deciding the contents of Deliveries:

- What will generate optimum feedback
- Delivering to *eagerly waiting* Stakeholders (otherwise, we won't get optimum feedback)
- Delivering the *juiciest*, most important Stakeholder values that can be made at the least cost, to raise the Stakeholder's interest to provide optimum feedback
- What will make Stakeholders more productive *now*

Also remember that:

- Every Delivery must have a useful set of values, otherwise the Stakeholders get stuck (for example, if there is a *Copy* function, there should also be a *Paste* function)
- Every Delivery must offer clear incremental value, otherwise the Stakeholders stop producing feedback
- Every Delivery delivers the smallest clear increment, to get the most rapid and frequent feedback

If the contents of a Delivery takes more than two weeks, it can be shortened: *try harder*

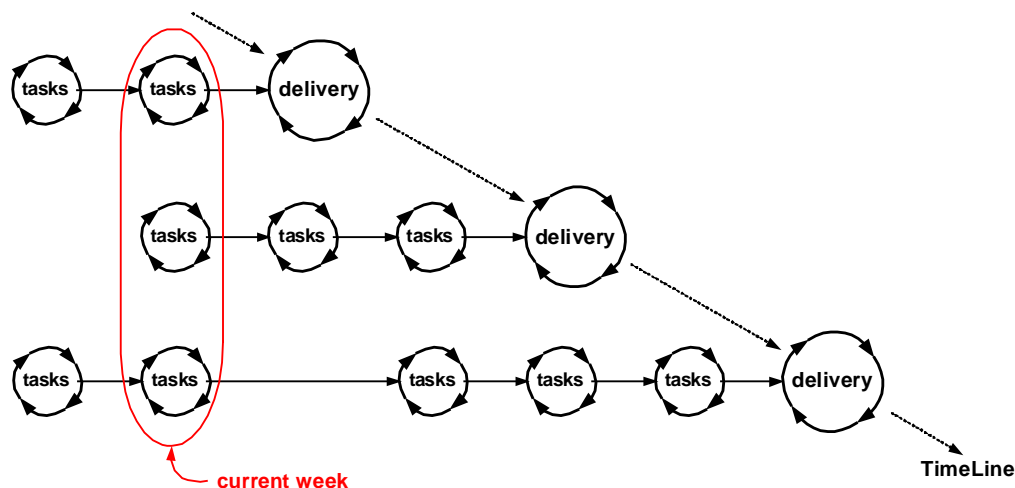


Figure 3: Tasks feed Deliveries on the TimeLine

5.5 TimeLine

We use the TimeLine technique to make sure that we will be on time (or even early). A TimeLine is a line between now and then. *Then* is any deadline (we also call it FatalDate): End of Task, End of Delivery, End of sub-project or milestone, or End of Project. A FatalDate is a commitment to deliver successfully, no excuses. We took the responsibility, so the Result will simply be there. If it is not, we failed to deliver on time. At the FatalDate, any excuse is pointless, because you *could have known before*. The moment you can foresee that, for whatever reason, you are not going to meet the FatalDate, you could have told the appropriate Stakeholders and we could have adapted our plans accordingly. Any day later you realize that you cannot meet the FatalDate, you have a day less to cope with it. If the time is up, there is no time left. You cannot change history.

During a project we constantly monitor where we are now, what the FatalDate is, and constantly optimize what we should and what we can do in between.

5.6 Tasks, Deliveries and TimeLine

Tasks feed Deliveries (see figure 3). Deliveries create focus for what to do in Tasks. In any TaskCycle we are working on the current Delivery. Because some Deliveries need more than two weeks to prepare we may also work on Tasks for future Deliveries. That said, we shouldn't start working on future Deliveries too soon, because the longer we work on a Delivery, the more the world may have changed, so that what we already did has become irrelevant. It really is a challenge to define Deliveries and to start working on the right Delivery, Just-in-Time.

On the TimeLine we are scheduling Deliveries in the best order to achieve the best Result in the least time. This is a dynamic process, because we may have to redefine Deliveries based on experience of the developers, feedback from Stakeholders, and market changes. We are constantly challenging the order of Deliveries to get the best route to the Result, with the fewest iterations. We may also have to change the order of Deliveries if somebody crucial for a Delivery is ill, or is needed temporarily on another project.

6 Evo practice

By collecting the experience of more than twenty-five projects between 2001 and 2004, we have arrived at several best practices that you can use to start new Evo projects.

These practices do not describe theoretical processes or how someone *thinks* we should work. They rather describe what works in day-to-day reality, where we have to cope with human psychological behavior that is not always as logical as we intuitively assume or might wish were true. In fact, Evo thrives on reality. Because of this, you can start using these practices tomorrow and immediately benefit. You don't have to call it Evo. Result is all that counts. That is never just "following process". Result is always measured as customer success at the least cost.

6.1 TaskCycle planning

At the start of the weekly TaskCycle, this is what we do:

1. Determine the number of hours you have available for this project this TaskCycle

People may work less than the full week. For example, they may take a vacation, follow a course, visit a dentist or work for more than one project. So we determine the number of available hours for this project first, because then we know when we can stop adding Tasks.

2. Divide this gross number of available hours into:

- **Available Plannable Hours (default 2/3 of gross available hours)**
- **Available Unplannable Hours (default 1/3 of gross available hours)**

We only plan those Tasks that don't get done unless planned. If you plan, you *have* time, and after that time, the Task will be done.

We do *not* plan Tasks that will get done anyway, even without planning. As a default ratio we start with 2/3 plannable and 1/3 unplannable time. In many projects this proves to be realistic. In a 40 hour work week, this means 27 hours plannable time, 13 hours unplannable time.

3. Define Tasks for this cycle, using the Task Selection Criteria

Focus on finding Tasks that are most important now and don't waste time on less urgent tasks for the moment. Based on what we learn from current tasks, the definition of later Tasks could change, so don't plan too far ahead. Use the Delivery definition to focus on what to work in the Tasks.

4. Estimate the number of effort hours needed to completely accomplish each Task

We always estimate *effort* hours. Ask people to estimate in days, and they come up with lead time (the time between starting and finishing the Task). Ask people to estimate in hours, and you'll find that they usually come up with effort (the *net* time needed for completing the Task). The reason for keeping effort and lead time separate is that the causes of variation are different: If effort is incorrectly estimated, it's a *complexity* assessment issue. If there is less time than planned, it's a *time-management* issue. Keeping these separate enables us to learn.

Only the person who is going to do the Task is allowed to define the duration of the Task. Others may not even hint, because this influences the estimator psychologically. If others do not agree with the estimation, they may only challenge the (perceived) contents of the Task, never the estimated time itself. Ultimately, when we agree on the requirements of the Task, the implementer decides how much time he is going to need; otherwise there will be no commitment to succeed.

5. Split Tasks of more than about 6 hours into smaller Tasks

We split the work into manageable portions. Estimation is not an exact science, so there will be some variation in the estimates. We are not bound by the exact estimated effort hours. We are only bound by the Result: at the end of the week, all committed work is done. If one task takes a bit more and the other a bit less, who cares? If you have several tasks to do, the variations can cancel out. If you have a massive task of 27 hours, it is more difficult to estimate and the averaging trick cannot save you any more.

6. Fill the available plannable hours with the most important Tasks

Never select less important Tasks. *Always* fill the available plannable hours completely.

7. Ascertain that indeed these are the most important Tasks to do and that you are confident that the work can be done in the estimated time

- Any doubt undermines your commitment, so make sure you can deliver.
- Acknowledge that by accepting the list of tasks for this cycle means accepting the responsibility towards yourself and your team, and that these tasks will be done, completely done, at the end of the cycle.

At this point, you will have a list of Tasks that will get done. If you cannot accept the consequence that some other Tasks will *not* be done, do something! You could:

- Reconsider the priorities.
- Get additional help to do some of the Tasks for you. Beware, however, that it may cost some time to transfer the Task to somebody else. If you don't plan this time, you won't have time.
- If no alternative is possible, accept reality. Hoping that the impossible will happen will only postpone the inevitable. The later you choose to do something about it, the less time you have left to do it. Don't be an ostrich: in Evo we take our head out of the sand and actively confront the challenges.

Evo Task Administrator tool

In all the projects coached since 2002, we introduced the Evo Task Administrator, or ETA tool, which is used to administer the Tasks. This MS-Access application can be downloaded free from www.malotaux.nl/nrm/Evo/ETAF.htm, together with an explanatory text. A screen shot is shown in figure 4.

The screenshot shows the Evo Task Administrator V1.12 application window. The window has a menu bar (File, Edit, Insert, Records, Window, Help) and a toolbar. The main form is divided into several sections:

- Left Panel:** Contains dropdown menus for Project (Dino-QUA), Delivery (1), TaskCycle (1), TaskType (Code), Priority (5), Who (Simon), and hrs (5). It also has a checkbox for 'Other work' and a 'Plan Reviewer' dropdown (Ale). A 'done (Checks)' section shows 'OK' and '100% done'. A 'Hours of Simon in Cycle' section shows '15 total', '15 OK', and '0 not OK'.
- Task Sheet Section:** Includes fields for Task Name (SRS Review), Cycle (1), Task cycle due date (19 mrt 2003 wk 12), Delivery Nr (1), Delivery Name (Delivery 1), and Delivery Due (2 apr 2003 wk 14). It also has a checkbox for 'Other work'.
- Task Description Section:** Includes fields for Task Description, Functional Requirements (what the result of this task should be), Performance Requirements (how well the result should do the what), Constraints (what not), Validation (how to check that the requirements are met), Implementation Ideas (solution direction ideas), Planning (to make sure task is done on time), and Unclears (anything that is still unclear).
- Table:** A table at the bottom lists tasks with columns: ID, Project, Delivery, Cycle, Task cycle due date, Pri, Who, hrs, Done, and TaskName. The table contains 20 rows of data.

ID	Project	Delivery	Cycle	Task cycle due date	Pri	Who	hrs	Done	TaskName
10	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Harry	5	OK	SRS Review
11	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Niko	5	OK	SRS Review
12	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Simon	5	OK	SRS Review
13	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Arian	5	OK	SRS Review
14	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Ronald	8	OK	SRS met stakeholders
15	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Ronald	2	OK	Fortran probleem + analyse meetwaarden probleem
16	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Harry	8	OK	Gui test tbv beslissing wel of niet aut gui testen
17	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Harry	4	OK	Raamwerk2: Deployment
18	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Niko	8	OK	Raamwerk2: Resultaat Arian: beheersgedeelte
19	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Harry	2	OK	Inrichten eigen systeem als Arian
20	Dino-QUA	Delivery 1	1	19 mrt 2003 wk 12	5	Niko	2	OK	Inrichten eigen systeem als Arian

Figure 4: Evo Task Administrator tool screen shot

6.3 TaskSheet

We use the TaskSheet to define what “completely accomplished” means. It helps us to check whether we are going to do exactly what is needed at this moment, not less and not more.

On the TaskSheet we can document:

- The requirements of the Task (Functional: *what*, Quality: *how well*, Constraints: *what not*)
- Task validation: how we are going to establish that the Task’s requirements are met
- The strategy to succeed this Task (planning within the Task, design approach)
- Whatever is still unclear

Before starting with the “real” Task, we ask the Project Manager, the Architect, or a colleague to review the TaskSheet. This may take only a few minutes, but it can also take more time. The longer it takes, the more important the review. Most reviews lead to changes in the TaskSheet. That’s nice, because we will be working more on the right things than we would have otherwise.

After the definition of the Task has been changed, or better defined, the Task time estimate should be reconsidered by the person who is going to execute the Task.

You might be concerned that the TaskSheet takes extra time. Using the TaskSheet doesn't cost time. It *saves* time. Try before you decide. If it ever proves to cost you time, find out why and act accordingly.

Note: If "completely accomplished" is defined as "first half of larger task finished", the TaskSheet should indicate how "first half finished" can be established. Don't settle for weak, un-measurable outcomes.

In the Evo Task Administrator (ETA) tool we have incorporated the TaskSheet for each Task, as shown in figure 4.

TimeBox

The number of effort hours planned for a Task is a TimeBox: this is the time available for finishing the Task *completely, no need to think about it any more*. If a Task proves to need more time than anticipated, don't just use more time:

- People tend to do more than necessary, so we may be able to do less without doing too little. The better the requirements of the Task are defined, the more focused you can go straight for the goal. That's why we use the TaskSheet.
- If you really cannot finish your task within the TimeBox, first complete the other Tasks. These were also chosen to have the highest priority: others may be waiting for their results.
- If you have time left after all other Tasks are done, you may still try to complete the Task.
- If the Task really cannot be finished, check:
 - What did you do
 - What did you not yet do
 - What do you still have to do

Then define new Tasks with estimations. These new Tasks may be considered in subsequent cycles. If the immediate continuation of the Task really seems to be more important than anything else: use the InterruptProcedure (see below).

Never decide *alone* that you can use more time than the TimeBox. As soon as you find out that the Task is going to need more time than you have available, discuss with the Project Manager: We decided to do this Task, based on the expected outcome (Result) against the expected estimation (cost). If the Task turns out to cost much more than expected, will the investment still be worth it? We might not even have started the Task, so the moment you find out, reconsider the priority: *don't just go on*.

6.5 At the end of the Cycle we Check, Act and Plan:

1. Was all planned work really done? If a Task was not completed, we have to learn:

- **Was the time spent but the work not done?**

This is an effort estimation problem. Discuss what the causes may be and decide how to change your estimation habits.

- **Was the time not spent?**

This is a time management problem:

- Too much distraction
- Too much time spent on other (poorly-estimated) Tasks
- Too much time spent on unplanned Tasks.

Discuss what the causes may be and decide how to change your time management habits.

2. Conclude unfinished Tasks after having dealt with the consequences:

- Feed the disappointment of “failure” into your intuition mechanism for next time. This is why commitment is so important: only with commitment we can feel disappointment. We must use the right psychology to feed our intuition properly.
- Define new Tasks, with estimates, and put them on the Candidate Task List. They will surface in due time. If they do not surface immediately, we apparently stopped at the right time. This ensures that we first work on the most important things.
- Declare the Task finished after having taken the consequences: remember that you cannot work on this Task any more, as it is impossible to do anything in the past.

3. Now continue with planning the Tasks for the next cycle

6.6 Analysis Tasks

If it will take significant time to define or estimate Tasks, we define an Analysis Task. In such a Task we don't do anything, we just analyze what we may have to do. At the end of the Analysis Task we check:

- What we know now
- What we still do not know
- What we still have to know

Then we define new Tasks or Analysis Tasks with estimations. Analysis Tasks get a deliberately small TimeBox. After, say, 2 hours we probably know a lot more than before starting. So after the short timebox we can much better define new Tasks or even new Analysis Tasks. By using a deliberately short timebox, we avoid spending more time than necessary. Analysis Tasks allow us to explore Requirements or to explore new techniques: we don't just start, we rather first analyze.

6.7 Interrupt

We know that requirements may change at any time, but we try to keep them stable during the TaskCycle. Sometimes, however, there are interruptions during the TaskCycle. For example: what do you do when the boss comes in and asks you to paint his fence? Or what do you do when a customer of your previous project reports a bug? In Evo, we don't immediately do such things because it's the boss or a customer. We also don't immediately reject the request, because it could be more important than anything else we are doing. However, because interrupts usually seem more important than they may be, we must never decide to change the plan and execute the interrupt on our own. Always consult the Project Manager.

If a new task suddenly appears in the middle of a TaskCycle (we call this an Interrupt) we follow this procedure, based on the principle “We shall work only on planned Tasks”:

1. Define the expected Result of the new Task properly
2. Estimate the time needed to perform the new Task, to the level of detail needed
3. Consult the Project Manager, or if unavailable, a colleague. You *must* seek a second opinion.
4. Check the Task planning
5. Decide which of the planned Tasks are going to be sacrificed (up to the number of hours needed for the new Task)
6. Weigh the priorities of the new Task against the Tasks to be sacrificed
7. Decide which is more important
8. If the new Task is more important: replan accordingly
9. If the new Task is *not* more important, then do *not* replan and do *not* work on the new Task. Of course the new Task may be added to the Candidate Task List
10. Now we are still working on planned Tasks

Small interrupts don't need the InterruptProcedure, as long as they don't jeopardize the completion of all the planned Tasks. Because our life is full of small Interrupts (drinking coffee, going to the bathroom, telephone calls, helping each other, and much more), we reserve the unplannable time for these unplannable Interrupts. The InterruptProcedure itself may be handled as a small Interrupt. If it needs more time, define an Interrupt Analysis Task first.

I know this may seem rather formal and bureaucratic. The only reason why we accept the bureaucratic rule in this case is because Interrupts are a big risk for the project and must be handled as such.

6.8 TimeLine

TimeLine is simply a line from Now to Then. We all can apply TimeLine quite well if we have to catch a plane: We know when the plane leaves and count back the time for checking in, the time to go to the airport, the time to get dressed and eat. This leads us to how we have to set the alarm clock the night before to make sure we will catch the plane. We also know that as soon as we can predict that we are going to miss the plane, we can abort the process even before going to the airport: we know we will be late, so it's no use trying any more.

In projects it is not very different, other than that what happens between now and then is a bit more complicated and a bit less predictable.

We call this technique of making sure we will be on time "TimeLine". It can be used on any scale: on a project, on deliveries, on tasks, the technique is always same:

1. Define a deadline or FatalDate. It is better to start with the end: planning beyond the available time/money budget is useless, so we can stop quicker if we find out that what we have to do takes way more time than we have.
2. Write down whatever you have to accomplish
3. List in order of priority
4. Write the same down in logical groups of Results
5. List these groups in order of priority
6. Translate the groups into Tasks: what you have to do

7. Estimate the Tasks in hours of effort (estimate less urgent tasks in less detail: they will be done later and hence will probably differ from what you think now. Don't waste time on irrelevant detail)
8. Cut the most urgent Tasks into work-Tasks of ~6 hrs effort or less
9. Review the order of the list
10. Ask the team to add forgotten tasks and add effort estimates
11. Get consensus on large variations of estimates (use a Delphi process)
12. Add up the number of effort hours
13. Divide by the number of available effort hours: This is the first estimate of the duration

What the customer wants, he cannot afford

The estimate of the duration is usually way beyond the required duration. At least we know now:

- What, at the FatalDate surely *will* be done
- What will *not* be done
- What may be done (estimation is not an exact science)

We also made sure that we plan to work on the most important issues first and the bells and whistles last.

Now you can discuss this with your customer. If what is surely done is not sufficient for success, you better stop now, to avoid wasting time and money and to spend it on more profitable activities.

In the beginning, customers can follow your reasoning, but still want it all. Remember that they don't even exactly know what they really want, so "wanting it all" usually is a fallacy, although you'd better not say that.

What you can say is: "OK, we have two options: In a conventional project, at the fatal day, I would come to you and tell that we didn't make it. In *this* project, however, we have another option. We already *know*, so I can tell you *now* that we will not be able to make it and then we can discuss what we are going to do about it. Which option shall we choose?"

If you explain it carefully, the customer will, eventually, choose the latter option. He will grumble a bit the first few weeks. Soon, however, he will forget the whole issue, because what you deliver is what you promise. This enforces trust. Remember that many customers ask *more*, because they *expect* to get less. He also will get confident: He is getting deliveries way before he ever expected it. And he will recognize soon that what he asked was not what he needed, so why bother to getting it "all".

The very first encounter with a new customer you cannot use this method, telling the customer that he will not get it all. Your competitor will promise to deliver it all (which he won't, assuming that you are not less capable than your competitor), so you lose if you don't tell the same, just as you did yourself before using Evo. If, after you won the contract, you start working the Evo way, you will soon get the confidence of your customer and on the next project he will understand and only want to work with you.

6.9 Weekly 3-step procedure

Based on the experience gained, starting with the weekly team meetings we found in most projects, we arrived at a weekly 3-step process, which proves instrumental for the success of Evo implementation. In this process we minimize and optimize the time used for organizing the Evo planning.

The steps are:

1. Individual preparation

In this step the individual team members do what they can do alone:

- Conclude current tasks
- Determine what they think the most important Tasks are for the next week
- Estimate the time needed for these Tasks
- Determine how much time they will have available for the project the coming week

The Project Manager also prepares for his team what *he* thinks are the most important Tasks, what he thinks these Tasks may take (based on his own perception of the contents of each Task and the capabilities of the Individual) and how much time he needs from every person in the Team.

2. 1-on-1's: Modulation with and coaching by Project Management

In this step the individual team members meet individually (1-on-1) with Project Management (Project Manager and/or Architect). In this meeting we modulate on the results of the Individual preparations:

- We check the status and coach where people did not yet succeed in their intentions
- We compare what the Individual and the Project Management thought to be the most important Tasks. In case of differences, we discuss until we agree
- We check the feasibility of getting all these Tasks done, based on the estimations
- We iterate until we are satisfied with the set of Tasks for the next cycle, checking for real commitment. Now we have the work package for the coming cycle.

We use an LCD projector at every meeting, even at the 1-on-1's. Preferably we use a computer connected directly to the Intranet, so that we are using the actual files. This is to ensure that we all are looking at and talking about the same things. If people scribble on their own paper, they all scribble something different. The others don't see what you scribble and cannot correct you if you misunderstand something.

If there is no projector readily available for your project: buy one! The cost of these projectors nowadays should never be an obstacle: you will recover the cost in a very short time.

There is not just one scribe. People change place behind the computer depending on the subject or the document. If the Project Manager writes down the Task descriptions in the Task database (like the ETA tool), people watch more or less and easily accept what the Project Manager writes. As soon as people write down *their own* Task descriptions, you can see how they tune the words, really thinking of what the words mean. This enhances the commitment a lot. And the Project Manager can watch and discuss if what is typed is not the same as what's in his mind. And when we are connected to the Intranet, the Task database is immediately up to date and people can even immediately print their individual Task lists.

3. Team meeting: Synchronization with the group

In this step, usually at the end of the day, after all the 1-on-1's are concluded, we meet with the whole team. In this meeting we do those things we really need all the people for:

- While the Tasks are listed on the projection screen (as in figure 4), people read aloud their planned Tasks for the week. This leads to the synergy effect: People say: "If you are going to do that, we must discuss ...", or "You can't do that, because ...". Apparently we overlooked something. Now we can discuss what to do about it and change the plans accordingly. The gain is that we don't together *generate* the plans, we only have to *modulate*. This saves time.
- If something came up at a 1-on-1 which is important for the group to know, it can be discussed now. In conventional team meetings we regularly see that we discuss a lot over the first subject that pops up, leaving no time for the real important subject that happened to be mentioned later. In the Evo team meetings we select which subject is most important to discuss together.
- To learn and to socialize.

At every step of the process we try to minimize the number of people involved. First we added the 1-on-1's to the process. The aim was to relieve the team meeting from individual status reporting and from too detailed 1-on-1 discussions. We found, however, that these 1-on-1's easily took about one hour each. One Project Manager said: "Niels, with 6 people in my team, I can just manage in one day. But what would you do if there were 15 people in the team? I want these meetings to take not more than 30 minutes". Watching closely what was happening in the 1-on-1's, we saw that there was a lot of thinking and waiting: "What are you going to do the next cycle?" Pause for thinking. "What effort do you estimate for this Task?" Pause for thinking. "How much time do you have for the project this week?" "I don't know. I have to discuss with the Project Manager of the other project". Sigh. Why didn't you check *before* the meeting? Now we cannot decide!

This led to the Individual Preparation step, where people prepare these issues before the meeting. The result was that the 1-on-1's went from one hour to 20 minutes. That was much better than we expected. The reason is probably that now people come to the meeting much more prepared, needing even less time to get to the point.

Now having optimized the 1-on-1's, Project Managers invariably say that these 1-on-1's are one of the most powerful elements of the Evo approach.

Team meetings usually take not more than 20 minutes. Do we discuss less than before? No, we just discuss the right things effectively and efficiently.

Conclusion: How Quality is Assured by Evo

Deming said that quality cannot be tested into a product; it has to be designed in from the beginning. Aren't we doing just that? In Evo projects we define what Quality is and then we pursue the defined Quality, constantly optimizing based on what we learn along the way. All the Quality Assurance people need to do is guide and coach us, watching over our shoulder to ensure we stay disciplined. Not because we like discipline, but because we like success.

Originally prepared for the Annual Pacific Northwest Software Quality Conference, Portland, USA, 2004 - Version 1.0f - 24 Jan 2008

OpenUP –The Best of Two Worlds

Bjorn Gustafsson bjorn @ goodsoftware.ca
GOOD Software Inc, www.goodsoftware.ca

Abstract

Software organizations looking to adopt an iterative and incremental process have found themselves left with less than ideal options. While RUP, the IBM Rational Unified Process®, was the first mainstream iterative software process, its complexity and size makes it difficult to adopt. Agile processes like Scrum and XP, on the other hand, are leaner, but their different culture and lack of documentation often meet resistance from management.

This dilemma is perfect soil for the new OpenUP process which packages the best RUP and agile practices in a light-weight open source process framework. The result makes management happy, since they get a stable and well-defined governance process, is easy to adopt, and serves the team a smorgasbord of software best practices right in their web browsers.

This article gives an overview of OpenUP and explains how it relates to both RUP, from which it received its foundation, and agile methods, from which it incorporates their best practices.

Introduction

Software development today is radically different than it was only ten years ago, and the software process landscape is changing too. Since its creation ten years ago, RUP [11] has become the de facto process in many software organizations. With a knowledge base of thousands of pages it offers guidance to a wide range of industries and systems. However, RUP is also “complex” and can be difficult to implement in an organization because of its size, complexity and learning curve.

More recently, enabled and fuelled by new conditions and initiatives in our industry, the Agile Manifesto [2] established a philosophy in 2001 that brought some ground-breaking new ideas to the software process landscape, which materialized in methods like XP [3] and Scrum [4]. Even though replacing RUP never was the primary motivation, ‘agility’ seemed like a good idea to many, including those that struggled with RUP.

Now, a few years later, we know that agility is not the cure-all for our software process pains, and software teams still keep failing. “Being agile” requires a change of mindset and attitudes throughout the whole organization and not all organizations are ready for this cultural change.

Some common problems can be observed in troubled projects:

- **The project team doesn’t share a clear vision of how the system will appear to its users.** Without a clear vision of the final system, there is no guiding framework for the work in the project. The team’s analysts have no means to calibrate their requirements to the scope and effort of the project, which results in ill-fitting requirements statements; and the development team can not properly prioritize their work.
- **Requirements do not drive development work.** Some development cultures regard requirements as “incidental” input to the project only, and drive the development work based on other, internal and technical, conditions. This is commonly found in “silo” organizations where there are separate teams for requirements capture and development.

- **The system's architecture has not been articulated.** Although projects that only evolve and maintain existing systems may not need to pay much attention to architecture, there are many projects that do. As Dean Leffingwell [7] points out: “... *how much architecture a team needs depends in large part on what the team is building*”.
- **Plans are not connected to the engineering reality.** Plans are often created and maintained separately from the actual project work. We have all seen nebulous Gantt charts that project managers spent days or even weeks creating, with hundreds of line items in nifty breakdown structures, purportedly believed to bring the project to “completion” at some well-defined point in time. Of course, these plans become outdated even before they are pinned on the wall.
- **Risks are ignored.** All projects run the risk of building the wrong product or not being able to build the product as envisioned, yet very few projects acknowledge this uncertainty and actively try to reduce it.

Whether your process is RUP, agile, or something else, and whether you are a programmer, architect, designer, tester, analyst or manager, you may recognize these problems in your own project. If you do, you are not alone. In fact, the majority of software projects are still problem-riddled.

Why is this?

First and foremost, software development is complex matter and orchestrating dozens or more individuals to build a complete software system is down-right hard work.

Advertisement – Fast Software Configuration Management - Click on ad to reach advertiser web site

Perforce | Fast Software Configuration Management



Introducing Folder Diff,
a productivity feature of Perforce SCM.

Folder Diff is an interactive, side-by-side display for comparing the state of any two groups of files.

Use Folder Diff to quickly determine the differences between files in folders, branches, labels, or your local disk. This is especially useful when performing complex code merges.

And when you've been working offline, Folder Diff makes it easy to reconcile and catch up with the Perforce Server when you get back online.

Folder Diff is just one of the many productivity tools that come with the Perforce SCM System.

PERFORCE
SOFTWARE

Download a free copy of Perforce, no questions asked, from www.perforce.com. Free technical support is available throughout your evaluation.

Every project is unique and most projects have some parameters that just aren't "ideal" for their process or they have a less than ideal process.

Many software organizations ask "How can we fix these problems?"

Both RUP and agile methods certainly have the solutions. The problem with RUP, though, is that they can be hard to find and put to practice; the problem with agile methods is that their advice and guidance can be difficult to translate to a particular project situation since they are based on tacit knowledge and textbooks only.

With OpenUP the situation is different. It packs short and concise guidance into a small number of pages, which are always just a couple of clicks away. Adopting OpenUP takes you a long way towards solving these and other problems.

Introducing OpenUP and the Eclipse Process Framework

While OpenUP is the tangible process product, it is also part of the larger Eclipse Process Framework.

"The Eclipse Process Framework (EPF) aims at producing a customizable software process engineering framework, with exemplary process content and tools, supporting a broad variety of project types and development styles" [1]

EPF is an open-source initiative that was started in 2006 with contributions by IBM Rational of parts of their RUP content and technology. Since its inception the project has actively involved more than 20 companies, and the first release was made available in September of 2007. Despite being an Eclipse project, EPF can be used to create process descriptions for any type of development, including J2EE on Eclipse or .NET using Microsoft Visual Studio.

EPF, like other Eclipse projects, offers exemplary implementations of its two components:

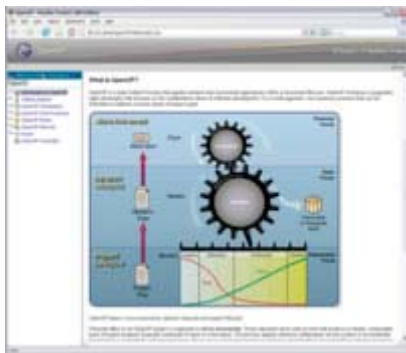


Figure 1: OpenUP process

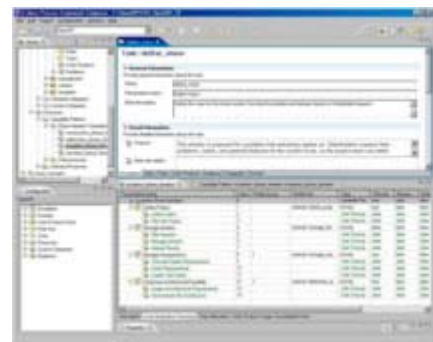


Figure 2: EPF Composer

OpenUP is a welcome addition to the software process landscape: it is agile, both in the guidance it provides and in "spirit", while at the same time being documented. As Per Kroll, the EPF project lead states *"the overarching goal with OpenUP was to create an agile approach to using RUP, while at the same time leverage all the good things we liked in other agile processes."* [6]

OpenUP borrows many strokes from RUP which it blends with agile practices, and the result is a complete process that suites many smaller-scale projects. It can also be extended for those projects and organizations that have more challenging circumstances.

Even if the OpenUP process is the focal point of EPF, it is only half the story. Much of its value lies in the fact that it is **open source** and that it is based on a **standardized meta-model**, called SPEM [5].

The advantages of open source are obvious: it can be used by organizations of all sizes at no cost. It also means that anyone can share best practices in the spirit of open source.

The advantage of being based on a standardized meta-model may not be as obvious, but is key to the overall success of EPF. Concretely, this makes OpenUP **extensible** and it can be augmented with any combination of standard, 3rd-party and proprietary practices. Just like UML provides a standard language for software design models, SPEM provides a standard language for definition and exchange of process models. With SPEM, process is expressed as a set of elements and components, and it is modular as opposed to a monolithic whole.

This is where EPF Composer comes into the picture. It provides the tools both to create new process content and to tailor OpenUP by selecting the desired set of components. It is a fully-featured process engineering tool and any organization can use it to develop their own process best practices, as well as share those with other EPF users.

This is the “grand vision” of EPF: to build an open source community around EPF, where industry best practices can be downloaded and exchanged. With this first release, this grand vision is starting to come into existence.

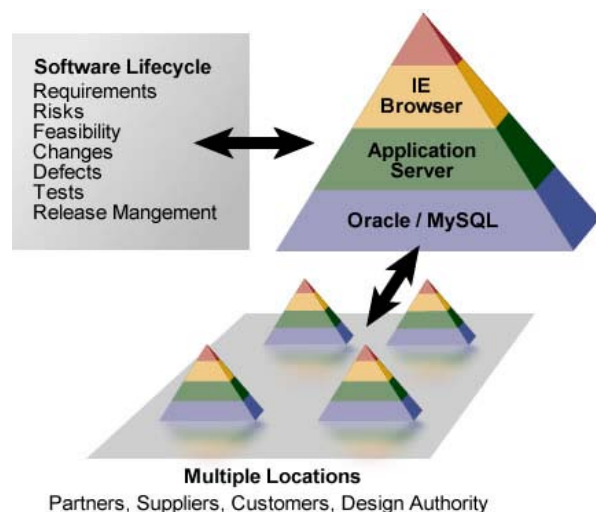
Advertisement – The Best in Requirements Management - Click on ad to reach advertiser web site

Looking for the best in Requirements Management, Engineering Information and Collaboration?

- Web-based interfaces
- Fast response time
- Wide-Area Network optimisation
- Multiple SQL database support
- Multi-platform support



PACE is a true enterprise solution to capture, analyse, develop, trace and manage information throughout the product lifecycle.



Let PACE™ do the Heavy
Lifting for you!

Visit [Http://Holagent.co.uk](http://Holagent.co.uk)

Telephone: +44 (0)14 9487 1915

OpenUP – the method

This chapter is intended to give a brief introduction to OpenUP only. There are excellent resources on the EPF web site [1] for learning more about it. (including downloading and browsing the OpenUP process itself!)

OpenUP defines a set of **roles**, **work products** and **tasks**:

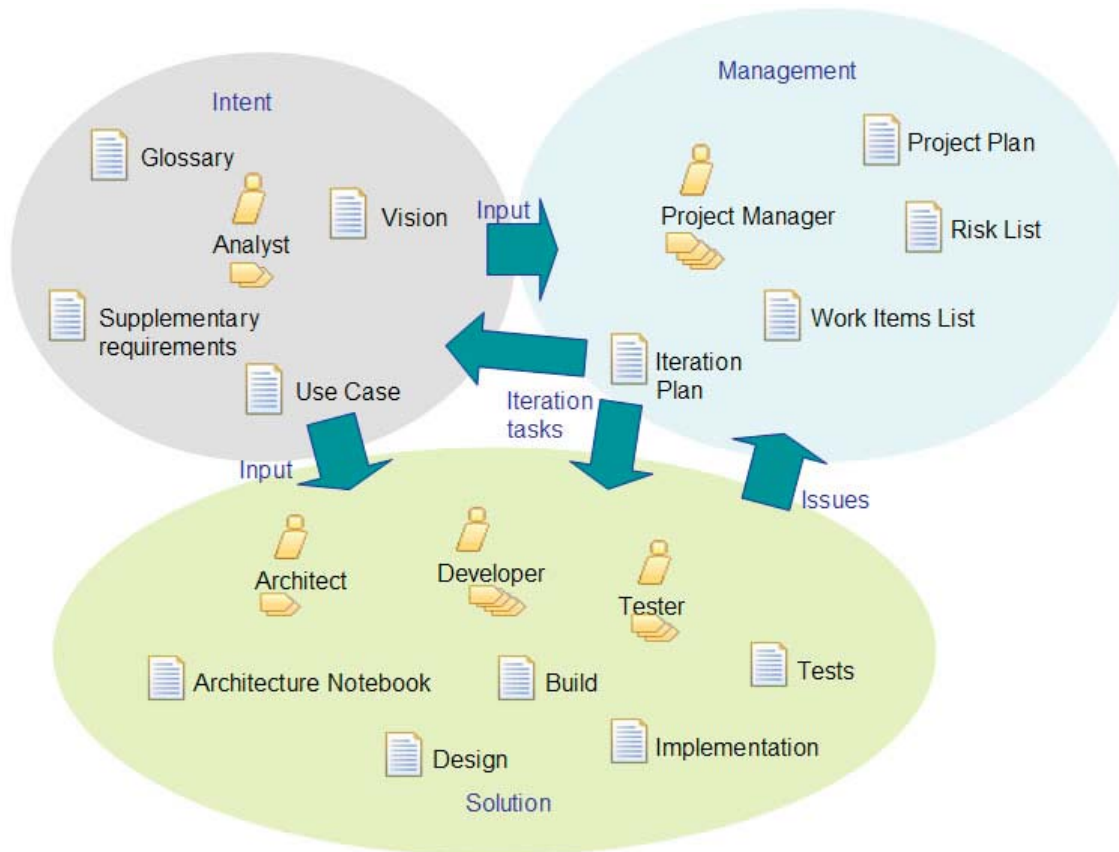


Figure 3: Overview of OpenUP elements

The process described by these elements is *minimal* and *complete*; it is the smallest set of elements that still describe a project end-to-end.

Analysts formulate the intent of the system in the vision, use cases and supplementary specification products.

Managing iterative projects is largely a matter of orchestrating the activities of the team through each iteration, primarily by managing tasks against the current iteration plan. It is populated with the highest prioritized items from the work items list and risk list, and the team commits to a certain amount of work in each iteration.

The development team includes architects, developers and testers, who are responsible for the development of the solution products, which ultimately result in the operational system.

Activity models describe typical task collaborations as they occur in iterations:



Figure 4: Example activity model

A lifecycle model provides the governance process for iterations and micro-increments:

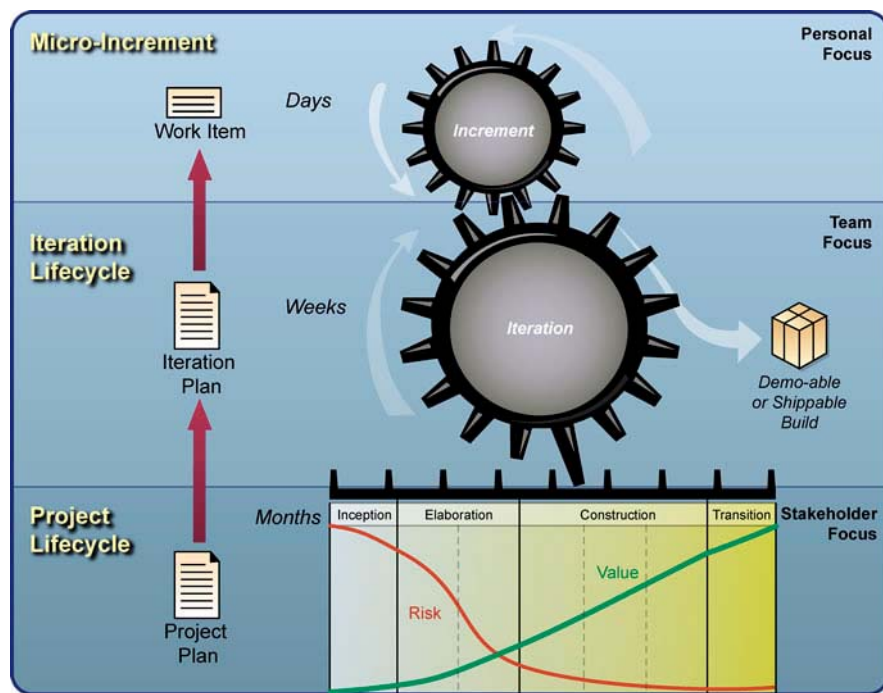


Figure 5: OpenUP lifecycle model

If you are a developer your process is the “analyze-code-test-integrate” cycle that you go through almost on a daily basis, as specified by the tasks in the iteration plan; if you are a manager, or Scrum Master, your focus is on how the team performs in each iteration, which is estimated and tracked in the iteration plan; and if you are a project stakeholder, you are focused on understanding what the project will deliver and when, as described in the project plan.

OpenUP is inherently *iterative and incremental* and the project is executed over a series of iterations, typically 2-6 weeks in duration.

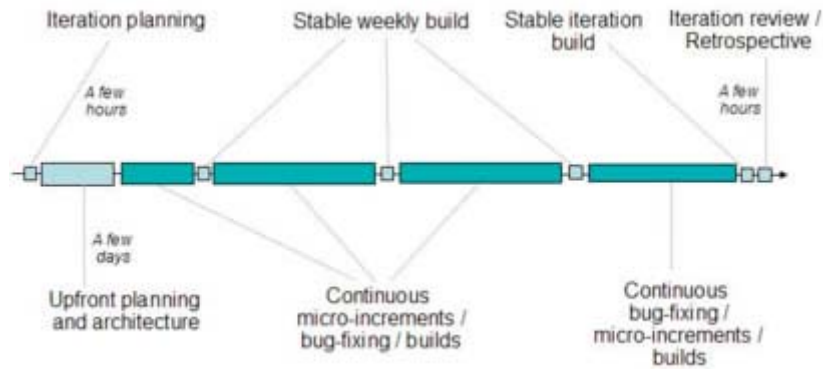


Figure 6: Iteration lifecycle

Each iteration takes on a subset of the project’s work items. Iterations are started with a short planning activity where the highest-prioritized items from the work items list and risk list are allocated to the iteration plan. This is followed by a short activity where the team gets involved in the detailed planning and estimation of each work item. Thereafter the iteration work commences and each feature is analyzed, designed, implemented, tested and integrated in its own micro-increment. As the iteration unfolds, task status is reported back to the iteration plan for overall project status.

The project lifecycle in Figure 5 identifies four distinct phases, each with a specific purpose and milestone criteria:

- **Inception:** define the scope and objectives of the project
- **Elaboration:** establish an understanding of the requirements and create an *executable architecture* for key scenarios and quality demands
- **Construction:** build the functionality of the system
- **Transition:** release the system to the end users

This lifecycle model distinguishes OpenUP from agile methods and allows us to focus early project efforts, in the inception and elaboration phases, on understanding the scope of the project and its solution before embarking on full-scale development in the construction phase. At the end of the elaboration phase we have typically spent only 20-25% of the total project budget over 30-40% of the project schedule.

With this brief introduction, let’s see how OpenUP helps address the specific problems that we identified earlier:

- **A shared vision is created in the inception phase.** The stakeholders’ key needs and features are captured in the Vision document. It describes high-level requirements and design constraints, and gives an overview of the system’s functional scope.
- **All project work is driven by use cases and other requirements.** The Work Items List constitutes a “laundry list” of features, requirements and change requests raised on the system.
- **An executable architecture is created in the elaboration phase.** The Architecture Notebook along with the other development work products represent a base-lined executable architecture that demonstrates how the system supports the key scenarios and constraints, and which serves as basis for the ensuing construction phase. It is worth noting that ‘architecture’ is not a separate “thing” – it is the underlying organization and qualities of the system being built – and the executable architecture is just a state of the underlying design, implementation and test products. To create the right focus on the architectural concerns we

prioritize those use cases and other requirements during the elaboration phase that involve the highest technical risks, as identified in the risk list.

- **Each iteration is planned “just in time”, and the project team is involved in the detailed estimation and planning activities.** For each iteration the Iteration Plan is populated with the highest-prioritized work items from the work items list and risk list. The team is responsible for identifying and estimating tasks for each work item.
- **Risks are pro-actively identified and mitigated.** The Risk List identifies a prioritized list of risks that are associated with the project. All critical risks have been removed at the end of the elaboration phase.

OpenUP thus helps us remove some of the main obstacles to project success early in the project and with only a small investment. The inception and elaboration phases also help establish and organize the key project work products and processes, so that construction can commence with a growing team and aggressive timelines with minimum friction.

OpenUP is a web-based knowledge base

OpenUP is installed in your team’s intranet and accessed using regular web browsers. It can be installed directly from the EPF website or, in the case you wish to create your own tailored variant, with the help of EPF Composer. If you are familiar with RUP, you will recognize OpenUP’s content browser and some of its content. If you are not familiar with RUP, you will find that OpenUP’s two-pane view and content are easy to understand and navigate:

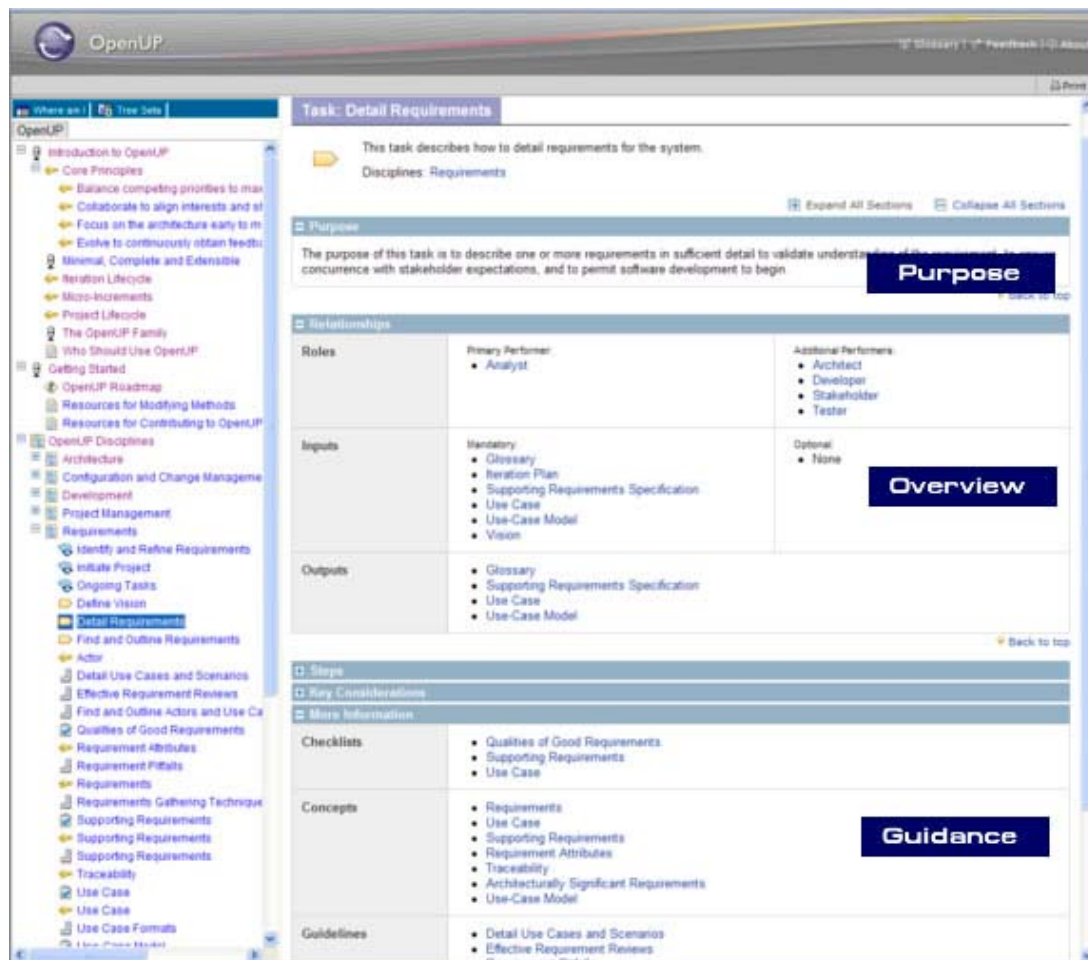


Figure 7: A sample OpenUP page

Figure 7 shows a *task* description and demonstrates the general layout of pages in OpenUP. Each element (*task*, *role*, *work product*) has a short description and links to its associated elements.

Each element has *guidance* attached to it, which further explains and facilitates their use in the project:

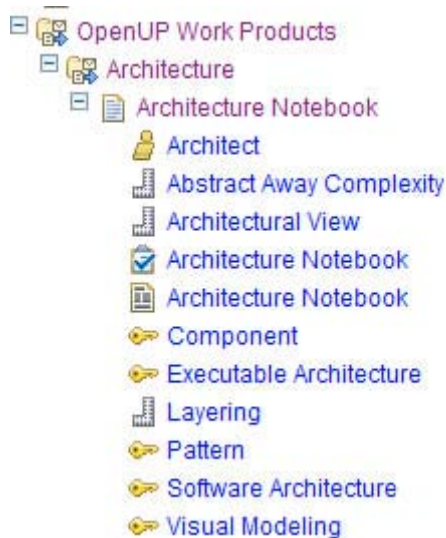


Figure 8: Example of guidance

The process web site provides a complete encyclopedia of your software process, and to understand OpenUP is a simple matter of browsing the OpenUP web site and studying those areas that are of immediate concern for the role you are playing in your project and the work products that you are responsible for.

Tailoring OpenUP to your needs

OpenUP can be tailored and extended using the EPF Composer tool:

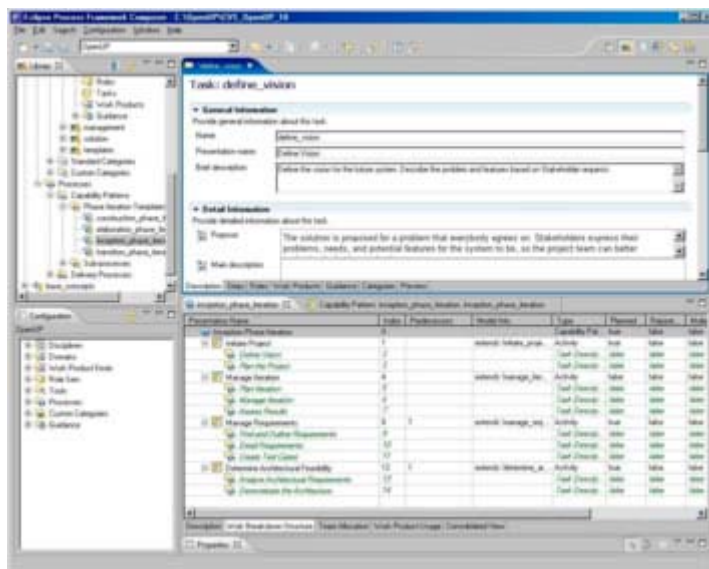


Figure 9: EPF Composer

It is centered around a form-based editor which allows you to quickly create and modify content. Your content can be as simple as a couple of additional guidance pages, or it can be as complex as adding completely new disciplines with new roles, tasks, work products and activities. You can even build an entirely new process family and ignore the existing OpenUP content completely. (An interesting example of this is the Scrum process that was developed as part of the EPF work: it uses the EPF process browser but is completely tailored to Scrum terminology and concepts)

A simple checkbox interface allows you to integrate your extensions with other OpenUP components in your own OpenUP variant.

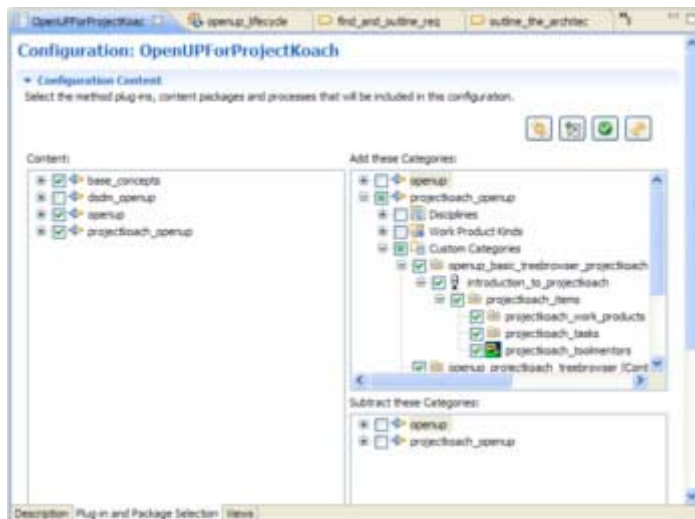


Figure 10: Checkbox selection of OpenUP configuration

Working with OpenUP

Normally, adopting a new software process is a major undertaking, and the transformation of the project team or organization never happens over night. Once a process has been implemented and used you want to find ways to improve it and make the organization more efficient. Process implementation and improvement involves understanding the process, defining it, and training the team on its practices, and this can take weeks, months or even years.

With OpenUP this is not the case. If you want to use it as-is - which is a good starting point for any project - you just download it from the web site and install it in your environment. It is self-contained and provides all the guidance you need to get started. Should you wish to improve the process from there, for example if your project is using a new technology and you want to add guidance specific to that technology, you will find it easy and straight-forward with EPF Composer.

OpenUP has many similarities with RUP, but choosing it is an either-or decision for RUP projects since they can not really co-exist in the same environment (and it wouldn't make sense because of their overlap). A goal with OpenUP was to create an agile process with focus only on the core software development element. This makes it manageable and easy to grasp.

Adopting OpenUP in agile projects, however, is not an either-or decision. It was designed to fit seamlessly with the work habits of agile projects. As we have seen, it balances agility with "just enough" governance, especially in the early stages of the project, to help establish the project context, fundamental structures and key principles early. Once status quo is reached – that is,

when the system has taken shape and the team has found a comfortable iteration pace – OpenUP is no different than most agile methods.

OpenUP is based on four core principles, all of which have direct correspondence with the Agile Manifesto:

OpenUP Key Principle	Agile Manifesto
Collaborate to align interests and share understanding	Individuals and interactions over process and tools
Evolve to continuously obtain feedback and improve	Responding to change over following a plan
Balance competing priorities to maximize stakeholder value	Customer collaboration over contract negotiation
Focus on articulating the architecture	Working software over comprehensive documentation

If you are used to working in agile projects you will find that OpenUP is no different when it comes to its underlying values and principles. Nor does it impose a different work style for your daily activities: you still do the daily stand-up meeting; iterations are planned the same way; you design, implement and test in small increments.

If you are a Scrum Master you will find striking similarities between the Scrum product backlog and OpenUP's work items list, and between the sprint backlog and the iteration plan. This is no coincidence since the inspiration for those came from Scrum. They are even used the same way in planning and performing iterations. (and – you can have your 30 day iterations too!) If you are a RUP project manager you will find that the process around the work items list and iteration plan is tangible and easy to manage.

OpenUP uses 'use cases' as envelope for requirements, but that doesn't preclude the use of user stories as a means to solicit feedback on the evolving system. Use cases and user stories are quite similar in nature, although use cases are coarser-grained than user stories are normally, and they come about in a more pro-active way.

As we have already seen, OpenUP acknowledges the value of architecture and regards it as an intentional property of the system. However, it doesn't treat it as a "large big up-front design" of the whole system. On the contrary, the activity of establishing the architecture is light-weight and focused on a small, central subset of the system's requirements and main constraints and objectives. This, too, is an iterative and incremental activity that occurs in each iteration, primarily during the inception and elaboration phases, in parallel with other project activities.

Of course, the approach to architecture taken by a particular project shall be determined by the circumstance of that project – the more complex and critical the project is, the more important becomes the architecture, and the more pro-active should the formulation of the architecture be.

The future of OpenUP

Like many other open source projects, OpenUP evolve in the direction set by the user community, so what the longer-term future has in stock for us is unclear.

We can rest assured though that, contrary to RUP and other proprietary processes, OpenUP will grow to include the practices most useful to the larger number of people in the software community.

Short-term, taking a quick peek behind the "development curtain", we see work currently being done to add the concept of 'practices' as primary building block, to replace the coarser-grained 'method plug-in' concept. This directly benefits all stakeholders of EPF and OpenUP, and ultimately serves you, the project member, with a more to-the-point process.

Other initiatives are integrating Wiki technology into EPF, to make augmenting and modifying process even easier.

Even if EPF is still young there are already integrations with project management tools available:

- Iris, Osellus Inc [10] (commercial)
- ProjectKoach, GOOD Software Inc [8] (free)
- Wilos project [12] (open source)

Also worth noting is that the future IBM Jazz [9] project incorporates EPF as one of its core components.

Summary

The new OpenUP process synthesizes the best practices from RUP and Agile methods into a light-weight and agile alternative to both. Given its RUP roots, it provides a process that is iterative and incremental, use-case driven, risk-driven, and architecture-centric, which at the same time supports the work habits of agile projects.

The OpenUP process features practices suitable for many projects right out-of-the-box, but it also provides the basis for adding 3rd party and proprietary practice extensions, using the EPF Composer tool, to tailor the most appropriate process for each project.

If you haven't looked at OpenUP yet, I invite you to take a closer look at the EPF website [1] where you find more information and downloads.

References

- [1] Eclipse Process Framework main web site www.eclipse.org/epf
- [2] Agile Manifesto <http://agilemanifesto.org/>
- [3] XP Extreme Programming <http://www.extremeprogramming.org/>
- [4] Ken Schwaber and Mike Beedle *Agile Software Development with SCRUM*, Prentice Hall 2001
- [5] SPEM specification <http://www.omg.org/cgi-bin/doc?ptc/07-11-01>
- [6] Per Kroll *OpenUP in a nutshell*
<http://www.ibm.com/developerworks/rational/library/sep07/kroll/index.html>
- [7] Dean Leffingwell *Scaling Software Agility*, Addison-Wesley 2007
<http://www.leffingwell.org/ssa.html>
- [8] ProjectKoach project management tool www.projectkoach.com
(GOOD Software Inc)
- [9] <https://jazz.net>
- [10] Iris process enactment tool www.osellus.com (Osellus Inc)
- [11] Philippe Kruchten *The Rational Unified Process, an introduction* 2nd Ed. Addison-Wesley 2000
- [12] Wilos open source project (<http://www.wilos-project.org>)

Real Reuse for Requirements

Doug Akers, doug.akers@mks.com
MKS Inc., www.mks.com

Introduction

A telecommunications company in a hotly competitive market needs to deliver the next generation of cell phone to its customers quickly, and at the lowest possible cost. The company wants to adopt a baseline set of requirements for the next generation project, but must make necessary modifications to leap ahead of the competition.

An automotive supplier must produce embedded software components consistently and reliably for its OEM clients. To do so, the supplier's development process must account for the slight variations required by each manufacturer.

Requirements reuse provides organizations, like those illustrated in the scenarios above, with the unique ability to share a requirement across projects without absorbing unnecessary duplication of artifacts within a repository. This is a critical capability that accelerates time to market and cuts development costs. Shared requirements can either track to the ongoing change made by the author or they can remain static if the needs of the project dictate. Further, change to a shared requirement can be made by anyone and the system handles the branching and evolution of that requirement appropriately.

The concept of reuse is a familiar notion within the software development realm, but less common when considered in the field of requirements management. There are various definitions and use cases which must be taken into consideration when implementing a solution to address requirements reuse.

This article discusses the elements that make up a requirement and establishes common understanding of how requirements evolve, how that evolution is retained, and how organizations can reuse requirements to speed business innovation, reduce complexity and control costs.

Dissecting a Requirement

To understand the concept of requirements reuse, we must first look at the various parts of a requirement: data, metadata and relationships.

Data

Describes an object, and is relevant to the object itself. An example of data may be a summary or description of a requirement.

Metadata

This is data about the data, which aids in organizing or using the object within a process. It typically describes the current state of the object, and has the same scope as the data itself. For instance, metadata may describe the State/Stage within a requirement workflow (i.e., Approved, Rejected, Satisfied, and Tested).

Relationships

This characteristic of a requirement allows you to model:

- structure (i.e., Consists Of, Includes);
- history (i.e., Revision Of, Derived From);
- conceptual links or traces (i.e., Satisfies);
- references (i.e., Defined By, Decomposes To);
- security (i.e., Authorized By, Enables).

Any given requirement can have information in each of the data, metadata and relationships categories. When requirements are reused, any or all of the information can also be reused.

An organization's chosen requirements management tool needs to have an underlying architecture and the user capabilities that support the strategic level of reuse dictated by the demands of the organization. Since reuse can occur at a number of different levels by leveraging the data, metadata and relationship elements of a requirement, flexibility is also critical to solving the reuse challenge.

History, Versions and Baselines

When implementing a complex reuse scenario, or even a system where requirements persist release after release, one must be able to identify significant points in that requirement's evolution. In the development world, these significant points are called "versions."

Advertisement – Software Test & Performance Conference - Click on ad to reach advertiser web site

Software Test & Performance
CONFERENCE
SPRING
BREAK OUT
OF THE BOX
A BZ Media Event

April 15-17, 2008
San Mateo Marriott
San Mateo, CA

BREAK YOUR OLD TESTING HABITS
Learn the Latest Tips and Techniques—
Try Out the Newest Technology—All at STPCon!
Register at www.stpcon.com

This term may mean different things to different people, so we will begin with a definition of the term “version” as it applies to requirements reuse and show how it relates to similar terms like history, baselines and milestones.

Consider a system where requirements are captured within requirements documents but are stored as individual items within the repository.

History is the term used to describe the audit trail for an individual item or requirement. All changes made to the item, whether it is to data, metadata or its relationships are captured in its history. History answers to the who, when and what questions with respect to changes to that item.

Version represents a meaningful point in an individual item’s history. Not all changes to an item are significant and warrant a new version of the item. For example, the reassignment of a requirement from Nigel to Julia would not require a specific version identifier. The change is recorded to the item’s history, but a new version is not created.

Baseline is a very similar concept to version but has a much different scope. Individual items are often organized into groups or sets. In the requirements management domain these sets are called documents and a baseline is a meaningful point in a document’s history. Some organizations use a slightly different definition for baseline. Rather than being a snapshot in time for a given document, a baseline, as defined here in the context of requirements reuse, is a goal to work towards. For the purposes of this discussion we will call the goal-oriented baseline a **milestone** in order to distinguish between the two.

Requirements management claims to allow for the versioning of individual requirements. Many tools support versioning by way of *cloning* or *copying* the entire requirement. Even fewer solutions relate the copy to the original requirement.

Although related, versioning and reuse are not the same. The concepts of versioning are often confused with that of reuse. In the next section, we will explore various reuse scenarios to illustrate the differences (and the benefits) of versioning and reuse.

Reuse or Not Reuse? – The Many Flavors of Requirements Reuse

Requirements Reuse without Reuse – Share

The ability to share an item between projects, documents or other work efforts could be considered a form of reuse. Under this definition all of the projects that are sharing the item see, and can possibly even contribute to, the evolution of the item. The metadata on the item is shared as are all the relationships and the data.

This is not real reuse. I question whether to call this reuse at all, but it is included here for completeness.

Requirements Reuse without Heritage – Copy

As mentioned previously, copying an object from one place to another can also be considered a form of reuse. In fact, this is the form of reuse that Microsoft Word (or any other non-Requirements Management tool) supports.

Advertisement – StarEast Software Testing Conference - Click on ad to reach advertiser web site

SOFTWARE TESTING ANALYSIS & REVIEW

The Greatest Software Testing Conference on Earth



MAY 5-9, 2008

Keynotes by International Experts



James Whittaker
Microsoft



**Bharat Mediratta
and Antoine Picard**
Google



Elisabeth Hendrickson
Quality Tree
Software



Tom Wissink
Lockheed Martin



Hugh Thompson
People Security



John Fodeh
Hewlett-Packard

**99.7% of 2007 Attendees
Recommend STAREAST
to Others in the Industry**



www.sqe.com/stareast

REGISTER EARLY AND SAVE \$200!



When an analyst opens a document, selects some content and performs a copy/paste gesture into another document, they are reusing that content for a new purpose. This form of reuse has no knowledge of heritage or “where did I come from” and of course changes in one document have no impact on changes in the other. In fact, changes are completely independent and one document has no knowledge that change occurred in the other, let alone what the change might have been.

This is also not real reuse. Any flavor of reuse must minimally include a pointer to where the original content came from.

Requirements Reuse with Heritage

Given the above scenarios, let us assume you can answer the “where did I come from” question. Augmenting the copy with the pointer back to its origin provides several options for reuse. It is the manner in which this link is leveraged that will differentiate each of the following reuse models. Most RM tools available today have some notion of links or relationships – if not at the individual requirement level, at the document level. Document level links are better than nothing, but they are not very powerful. In the long run, they don’t really answer the traceability question in sufficient detail to be meaningful.

Having a link to an item’s origin is the start of real reuse though it is certainly not the end.

Requirements Reuse with Change Notification

In this situation, a requirement and all related information (data, metadata and relationships), is reused in its entirety. Project state determines the state of the requirements at the time of reuse, and any change to requirements in a reuse scenario causes a ripple effect, flagging all artifacts related to those requirements as suspect.

Requirements Reuse with Change Control

Reuse with Change Control is similar to Reuse with Change Notification in that data, metadata and relationships are reused in their entirety. This seems, and in fact is, the same as the Share topic discussed above, however, there is one significant difference; the two projects sharing the same requirement only share it until the point in time where one project needs to change it. When the information changes a new version/branch is created and only items referencing that new version are declared suspect. All other projects or documents are unaffected.

Requirements Reuse with Annotations

In the two reuse paradigms above, the requirements and related information (data, metadata, and relationships) are reused in their entirety. In Reuse with Annotations, only some of the information belonging to a requirement is identified as a candidate for sharing and reuse. The rest of the information is specific to the project or document. The shared information is held in the repository while the other information belongs to the project or document reference. Each instance of the requirement being reused has its own metadata and relationships. The project or document state is, or can be, independent of the state of the requirements that are contained within it. New versions of the requirement are automatically created when the shared information in the repository is changed. These changes that trigger new revisions can suspect other references, as well as other items in the system, by the ripple effect of that change. For example, changes to requirements may affect test cases or functional specifications downstream.

Once you have project or document independence in terms of the metadata, you have the ability to model both a dynamic (share) and static (reuse) form of reuse at the same time. The project manager or analyst decides if they want to remain consistent with the evolving requirement in a dynamic way or if they want to lock the requirement down such that the impact of change does not affect their project.

Requirements Reuse with Annotations and Change Management

Applying change and configuration management paradigms onto the requirements management discipline in a single integrated and traceable solution can bring the power of reuse to a new level. By incorporating a process on top of reuse and controlling how and when requirements can be modified and reused enables you to reap these benefits without unnecessarily branching and versioning objects unless it is authorized and appropriate to do so. Requests for Change (RFCs) come in, get filtered and are directed by various review boards. Some of these RFCs get approved and assigned to users to affect the requested changes. Ideally, this change management process can define what types of changes can be made; whether it is modification, branching, applying a baseline or other gestures. Only when an approved RFC is associated with a requirement can an analyst modify the requirement, causing the system to version and branch accordingly, and notifying the related constituents appropriately.

There are clearly, additional reuse models that are not described herein – Component level reuse, documents reuse and various combinations of these with annotations and change management for example. This paper provides only a sampling. The business needs and strategic goals within the group, business unit or business as a whole will help determine which model is most effective for the project or organization.

Is Requirements Reuse Right For Your Organization?

Requirements reuse is not for everyone. There is a broad spectrum of need in terms of requirements management tooling in the market today, and organizations first need to know where they lie on the requirements maturity curve. The requirements maturity curve is not really a curve at all but a measurement of the current process and tools used and/or needed within an organization when it comes to requirements management. As organizations evolve along the curve, the need for more capabilities – such as change management, process and workflow, traceability, reuse, etc. – within their requirements management framework exists.



Many companies are still in the infancy of requirements management. They have not yet adopted a requirements management tool, and are currently using business productivity applications such as Microsoft Word or Microsoft Excel to capture and track requirements. They may look for capabilities such as ease of document import, rich text support, and downstream traceability to ease business adoption. These organizations are not yet at a point of requirements sophistication where reuse support is necessary – or maybe they are but have not found a tool to support their needs.

However, if an organization has progressed on the maturity curve with respect to requirements management, and is managing multiple projects and thousands of requirements in parallel and seeking to reduce complexity, lower cost of development, and shorten innovation cycles, then requirements reuse is a concept that should be investigated.

Let's face it, regardless of where an organization falls on the curve, reuse in its most basic form will provide a boost to productivity. Rather than re-writing requirements, copy them and modify them for the needs of each project – you will save keystrokes as well as leverage the structure and organization these requirements were managed under in the past. After all, how many different ways can the requirements for logging in to an application be specified really? Ok, likely quite a number, but within any one organization the need to standardize and streamline application access exists and leveraging requirements from one application to another to provide this similar type of functionality can only be a good thing.

In any case, concentrate on the problem domain before jumping into the question “is requirements reuse right for me?” What challenges is the organization facing in terms of requirements management? Here is a list of sample questions an organization can ask to determine if reuse is a concept that could be leveraged and if it is, which flavor of reuse is best suited to the need.

- How do you know you have captured all the requirements from your customer and the business? (Authoring)
- How do your project teams leverage work done by other projects? (Reuse)
- How do you verify that each and every change made to software systems track directly to its business requirement? (Traceability)
- How do product variations get tracked? (Reuse)
- Have requirements changed, been removed or added in the last month? How can you tell? (Change Management)
- How do you assess the impact of changed requirements on development schedules and resourcing? (Impact Analysis)
- Can you assess the effect on other projects sharing and/or reusing these requirements? (Reuse)

Chances are good that any given organization will have difficulty answering more than one of the above questions and that being the case a well defined requirements management process with the support of a tool that embodies effective authoring, traceability, change management and reuse while also providing similar capabilities to manage downstream activities like test management and software change and configuration management will enable greater control and agility over software projects.

All that is left to do, as if it is as easy as that, is to determine the business value associated with solving these challenges and what priority do these solutions have within the organization. Reuse may not be part of the short term strategy but successful strategic companies invest in the future and that means a process and a tool framework that will grow with the organization as it matures over time.

Additional Resources

Software Product Lines – Reuse that Makes Business Sense

<http://www.sei.cmu.edu/productlines/ASWEC2006.pdf>

Requirements Reuse and Feature Interaction Management

http://www2.enel.ucalgary.ca/People/eberlein/publications/FI_ICSSEA2002.pdf

Requirements Evolution and Reuse Using the Systems Engineering Process Activities (SEPA)

<http://journals.sfu.ca/acs/index.php/ajis/article/view/294>

Webinar: Increasing the Agility of Your Software Organization with Requirements Reuse

<http://www.mks.com/mt-requirements-reuse>

Creating an Agile Environment

Gregory S. Smith. This article is based on a chapter from "Becoming Agile"

www.manning.com/smith

A few months ago I was contacted by a friend with a problem. The year was coming to an end and he had let a compliance project slip through the cracks. The compliance deadline was year end which was a mere 5 weeks away. Failure to comply could mean serious government repercussions to his company. My friend asked for help in creating an Agile team and doing an Agile project in the following 5 weeks.

This would be a great time to tout how Agile came in and saved the day but that would be a lie. I did help my friend prioritize his work and make the deadline, and we did follow some Agile principles along the way, but we did not put an Agile team or process in place.

Why didn't we put an Agile team in place and follow an Agile framework? Because it takes time. Teams need time to feel comfortable with Agile processes and they need time to learn how to interact with each other. Managers need time to learn how to lead in an Agile environment. The team needs to use an Agile process for several months, then major benefits will begin to manifest.

Migrating to Agile is more than changing your process. It also requires a change in culture. For most companies changing culture is the most difficult part. I believe this is true for several reasons. Here are a few:

- Whether successful or not, companies get comfortable with their processes.
- Many people still believe requirements change because they are poorly managed. They cannot comprehend a process that embraces change.
- Most managers have been trained to control events. Empowering the development team to deliver and own the project is not intuitive or logical.
- Job protection. In larger companies whole groups are dedicated to regulating and overseeing projects. An Agile team has less need for these services.

There are numerous other reasons but I believe these are at the center of the issue.

These issues should be addressed in two ways. First, you want to address the culture needs of each group head on. We will lay out a game plan for obtaining support from line management, the team, the individual and executive management.

If you work in a smaller company

If you are in a smaller company you may not have all of the organization levels discussed in this article. That is a good thing. You should find it easier to create an Agile culture because you are fighting your competition on a daily basis. You will obtain the most value by reading the sections related to creating an Agile team and addressing the needs of the individual.

Second, you want to address this problem by establishing practices that foster an Agile culture. Practices such as high customer involvement, testing early, and collaborative decision making will promote an Agile mentality throughout the company.

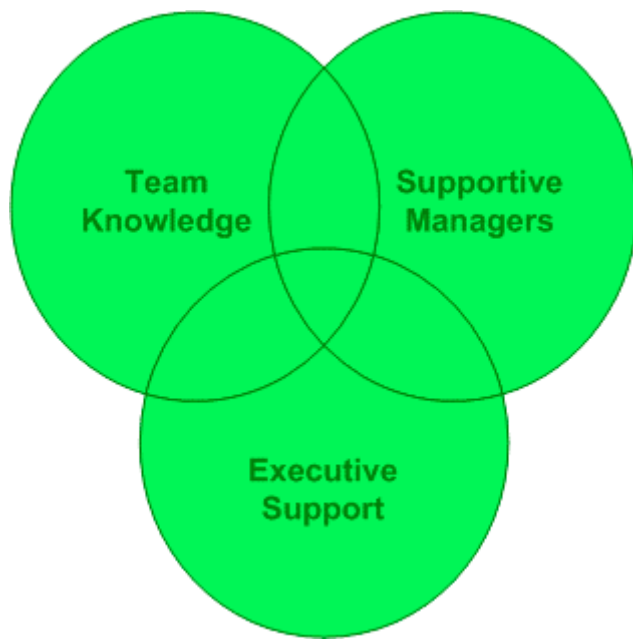


Figure 1. An Agile culture is established when the 3 major groups come together within a company. Executive management endorses the Agile principles, working managers learn to coach instead of direct, and the project team understands and supports Agile principles and practices.

This article will conclude by initiating our case study. We will get to know some of the people at Acme Media and see how they kick off their move to a more Agile process. We will create the Core Team and give them responsibility for learning Agile principles, then applying those principles to the tailoring of an Agile methodology for your Acme. The core team concept also helps with the cultural aspects of migration by involving project team members immediately.

The information in this article establishes the foundation that allows an Agile process to thrive. Similar to software development, if you get a good foundation in place it makes everything else easier to do. If you do not you will fight the foundation with every change you make. Let's look at the skills required of an Agile manager.

The Agile manager—more shepherding less directing

Do you remember a commercial for a company named BASF a few years ago? Their slogan was 'We don't make a lot of the products you buy. We make a lot of the products you buy better'. This is true of the Agile manager. An Agile manager will never write a line of code, never document any requirements, or test a feature. What an Agile manager will do is:

- Help the development team track true status
- Encourage the automation of redundant, repeatable tests
- Mentor the team on Agile processes and demonstrate the value.
- Help the team break the work into small chunks that can be delivered quickly.
- Ensure the work being delivered is in tune with the customer need.

An Agile manager provides leadership without using formal power. Instead the manager leverages the respect they earn from the team as they establish a history of working together to successful delivery of projects.

What does a manager need to do to establish a record of successful project delivery? Let's start with the soft skills.

The soft skills

If you look up “soft skills” on the United States Air Force website you will find, “A set of skills that influence how we interact with each other. It includes such abilities as effective communication, creativity, analytical thinking, diplomacy, flexibility, change-readiness, and problem solving, leadership, team building, and listening skills.”

This definition is an excellent prescription for the behaviors an Agile manager needs to subscribe to.

- Effective communication to ensure the team is synchronized on information.
- Analytical thinking to help the team brainstorm solutions when a challenge is encountered.
- Diplomacy skills to ensure tactful communications that do not offend or touch upon sensitivities.
- Great listening skills to not only ensure accurate understanding, but also to enhance relationships with others.

Advertisement – Agile 2008 Conference - Click on ad to reach advertiser web site



Expanding Agile Horizons

LEARN HOW TO DELIVER BUSINESS VALUE WITH AGILE

Join us in Toronto for the biggest ever gathering of agile practitioners and thought leaders. This conference expands Agile from software development to delivering business value. At Agile 2008 you will discover how to put agile principles to work!

Our program is organized like a music festival that provides different stages to attract audiences with common interests, such as tools, culture, leadership & teams and user experience. Each stage will have a feel of a smaller, focused mini-conference whilst providing you with wide choice of topics to choose from. At Agile2008 we provide a program suitable for all experience levels from novices to experts.

We are pleased to announce opening keynote speaker Jim Surowiecki, author of the book "Wisdom of Crowds". Bob Martin, author of "Agile Software Development, Principles, Patterns, and Practices," speaks at our last night conference banquet open to all attendees. Our closing keynote is Alan Cooper "Father of Visual Basic" and the inventor of using personas in Interaction Design.

Early bird registration is open now! Details at www.agile2008.org

 **Agile2008**
Conference

Toronto August 4 to 8, 2008

In summary, behaving in a way to enhance human relations.

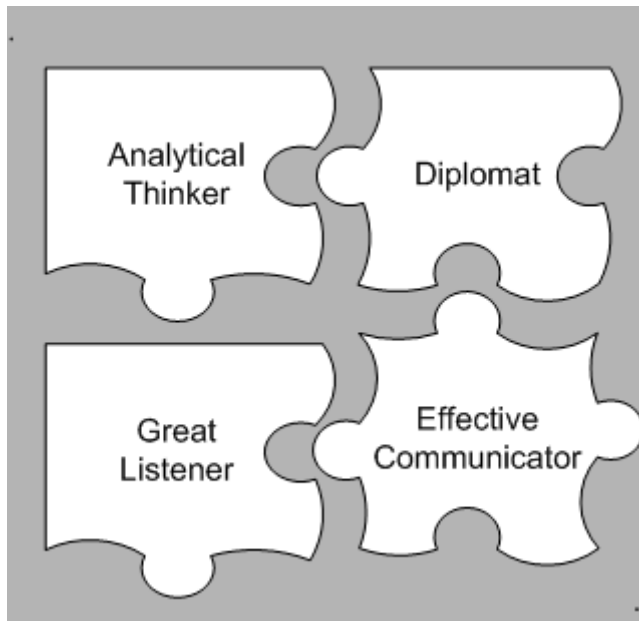


Figure 2. An Agile leader brings their soft skills together to “shepherd” the team versus directing them.

Diane Ehrlich, Ph. D in the Human Resource Development program at the University of Illinois defines soft skills as “The Skills needed to perform jobs where job requirements are defined in terms of expected outcomes, but the process(es) to achieve the outcomes may vary widely.” This is a good description for Agile development in general. You have a desired output, a project, and the way to achieve that output varies wildly depending on the specific needs of the project. Now let’s discuss where the soft skills are used.

Working with other managers

A project manager is usually leading a group of people that are not his or her direct reports. As mentioned before, your job is to earn the respect of team so they will follow you regardless of your express authority. In order to do this you also need to have the respect of the line managers who own the resources. The key is to ensure the line managers have bought into Agile concepts before you ask the team to.

Some level of training needs to occur within the line managers before an Agile migration is pursued. This training can come from any resource, internal or external, but during this training managers need to normalize on their support of the principles. You do not want to ask the manager’s directs to buy-in to the process before the managers have.

You also need to consider roles when working with other managers. Although everyone is flexible in the tasks they perform in an Agile environment, there will be areas of responsibility for everyone.

Consider the development team. The development manager usually acts as a technical mentor and also assigns tasks to the development team. Historically the development manager may have been in charge of reporting status for the development team.

In an Agile environment there is a 10 minute daily stand-up meeting for the whole team to discuss what they did, what they will do, and any roadblocks they have encountered. This meeting may be facilitated by the project manager or it may be facilitated by a development manager. These types of decisions should be worked out with managers in advance of deploying Agile.

Working with stakeholders

Another group that will be vital to your project success is the stakeholders. Stakeholders are defined as those who have interest or influence on the project. Typical stakeholders include senior management and indirect customers such as: support teams, maintenance teams, help desks, 3rd parties that integrate with the system, and other related product groups within the company.

In effect, stakeholders are another type of customer. They have their own needs that they want addressed by the project. To ensure successful delivery of your project you will need to integrate their requirements into a unified design.

All of the soft skills mentioned earlier are useful when working with stakeholders. The stakeholders may not be the main customers of the project, but you want them to feel valued. You want to demonstrate good listening skills and make sure they know that you understand their needs. You also need to demonstrate diplomacy and not upset the stakeholders by consciously providing information in a way that will inflame or incite them. For example, you do not want to demonstrate good listening skills and then immediately tell them that other stakeholders do not support their needs.

Demonstrate the value

The most important role of the Agile manager is to exemplify the Agile principles and live them daily. If you want the team to follow you, you must provide a strong example. There are numerous principles to emulate and follow. Here are the ones that provide the most impact.

“Just enough” planning

In traditional project management you identify features and then specify their requirements. Typically an analyst wants to answer every question possible in the specification so the development process will not be impeded by a missing requirement.

In Agile planning you want to plan “just enough”. Just enough planning to determine which features you want to build. Just enough coding to demonstrate the feature to the customer and verify that you are on track.

This is one of the hardest habits to break with a traditional team and the Agile manager needs to champion this mentality on a daily basis. The manager can also emulate this behavior by creating project plans the same way. A plan that has just enough information to get to the next level of the project, not a complete work breakdown structure before development has even begun.

Always ready to stop, drop, and deliver

Agile development is performed in iterations to enhance urgency and to support early delivery of the most valuable functionality. The project manager needs to infuse this mentality into the project team.

The project manager gets the team to put the same urgency around an iteration as they do with a project deployment deadline.

Unrelenting pursuit of customer value

An Agile manager is always thinking about the customer and their needs. All other measurements of a project are meaningless if the product delivered is of no use to the customer. There are 3 steps to ensuring the customer's needs are addressed:

1. Clearly define the customer(s). Many projects get underway with a light understanding of who their customer is. Make sure your customers are clearly defined and their specific needs are clear.
2. Develop a relationship with the customer. Get to know them well and integrate them into the project team. Use your soft skills to collaborate with the customer frequently and make sure they can be easily accessed by the team.
3. Be an advocate for the customer at all times. When features are being discussed and the customer is not present, put the customer hat on and envision what their response would be to the discussion. Share those thoughts with the team.

Ensuring technical excellence

The technical skill set of Agile managers vary. A manager can come from a classic PMI background, be a former developer, or have worked as a business analyst in the past. Regardless of the technical knowledge all Agile managers can push the team to pursue technical processes that embed Agile beliefs. Here are some of the best practices for obtaining technical excellence:

- *Create a process for continuous code integration.* As functionality is completed, developers integrate their work into the existing code base. The key is to integrate as small pieces of functionality are completed as opposed to waiting for a complete feature. This practice identifies code issues early and minimizing the complexity of tracing down issues.
- *Automate testing wherever possible.* Work with the team to automate testing wherever possible. This is usually easiest to do with regression testing. You can also automate daily smoke tests to speed up testing.
- *Perform a daily build/smoke test.* Related to automated testing, a daily build also helps mitigate risk by identifying code issues early. The daily test focuses on ensuring the critical pieces of the application are still functional.
- *Consider scalability.* As an application is being developed the team should consider future growth. What will happen if the application is extremely popular and usage exceeds expected volumes? The team can consider scalability as they design and ensure the design can be "scaled" easily if needed.

Conference Sponsor:

RALLY

SOFTWARE

KEYNOTES
BY INTERNATIONAL EXPERTS

Jean Tabaka

Rally Software Development

Johanna Rothman

Rothman Consulting Group, Inc.

Michael Mah

QSM Associates

Ken Schwaber

Advanced Development Methods, Inc.

The Venetian

**TUTORIALS
WORKSHOPS
CLASSES**

CELEBRATING 5 YEARS

Alan Shalloway	Julie Gardiner
Andy Glover	Ken Pugh
Andy Hunt	Ken Schwaber
Andy Kaufman	Kent McDonald
Ari Takanen	Kevin Bodie
Beth Layman	Lee Copeland
Bob Hartman	Lee Devin
Chris Ronak	Linda Rising
Chuck Allison	Linda Westfall
Dan North	Lisa Crispin
David Garms	Michael Mah
David Herron	Michele Sliger
David Spann	Mike Seavers
Ed Weller	Mike Tholfsen
Elle Ringham	Mitch Lacey
Guruprasad Gopalakrishnan	Nelson Perez
Herbert (Hugh) Thompson	Paco Hope
Hubert Smits	Payson Hall
James McCaffrey	Pollyanna Pixton
James Newkirk	Rebecca Wirfs-Brock
Jared Richardson	Richard Bender
Jean Tabaka	Rob Myers
Jeff Patton	Robert Galen
Jeff Payne	Stacia Broderick
Jerry Smith	Tim Korson
Jimmy Xu	Todd Little
Johanna Rothman	Vladimir Pavlov
John Janakiraman	Will McKnight

www.sqe.com/bsce

A great collaborator, communicator, and relationship builder

Agile is as much a team culture as it is a software development methodology. It is a culture of frequent conversation and consensus building. It is face to face interaction on a daily basis. The Agile manager needs to emulate the correct behavior to ensure positive results from the frequent meetings and daily decisions.

The correct behavior begins with checking your ego at the door when you walk into the office. There will be intense discussions in an Agile environment and the process will break down if you take criticism of your ideas as criticism of yourself. Demonstrate this to the team by never getting upset (at least visibly) during subject discussions that you are passionate about. Share your thoughts with passion, but expect and embrace criticism of your ideas. The team will adopt this behavior if you model it consistently.

Another key behavior is how to interact with those outside of the direct project team. These groups can be vendors, stakeholders, other departments, and sometimes customers. If you show respect in your interactions with these groups the team will too.

It is natural to talk poorly about others outside the team. “That stupid department we work with that does not do Agile”, “that idiot vendor whose system always goes down”, “that dumb customer who can never make up their mind”. You will always have this at some level at your work. The goal is to not let it get out of hand, where it exceeds venting and proceeds to a truly poor relationship.

Leading the team to ownership

In 1998 Arthur Andersen published a book by the name of "Best Practices, Building Your Business with Customer Focused Solutions". One of the best practices outlined in the book was the ABO Continuum. The continuum identified a vital element in introducing change to an organization, ensuring ownership of the change.

The continuum promotes the belief that organizational change goes through three steps; awareness, buy-in, and ultimately ownership.

In the awareness phase information about the change is shared early and informally. For example, during a team meeting a manager could say “the executives are discussing improvements for our development process”. The manager could also add in when he thinks he will hear more and see what the team reaction is.

The value is not so much in what is said, but when it is said. Every individual has their own timeframe for evaluating a change. The earlier you can make a group aware of a potential change the better your chances of getting them to “buy-into” the change when you are ready to roll it out.

The buy-in phase occurs when you roll-out the change and begin implementing it. Awareness has been created and you are looking for the team to consider the change and to use it with your guidance.

In the ownership phase the team has tried the change, begun to believe in it, and adopted it as a standard practice. They do not need management to encourage them to use it. They believe in it and will use it without being prodded.

The ABO Continuum is a great approach for rolling out an Agile methodology. Partner with your executive team during rollout and process to minimize pain in your migration.

“The Scrummaster”

Scrum has become one of the most popular Agile packaged methods. The Scrummaster is at the heart of Scrum. This individual is not a manager but more of a process facilitator and guide. A Scrummaster:

- Helps the team develop practices that support Agile principles
- Acts as a guide in training the team on how to be Agile and use Scrum
- Removes impediments that prevent the team from delivering software
- Shields the team from corporate bureaucracy and activities that do not add value to software development
- Champions engineering excellence and processes that support the creation of shippable software
- Ensures the team has direct access to the customer

I believe Scrum is a good Agile method, especially when there is urgency to establish a development process quickly in an immature organization. However, I struggle with the depth of responsibilities assigned to the Scrummaster.

My opinion is driven by something I learned when I became certified as a Scrummaster. My Scrum teacher told me that Scrummasters are the key to Scrum’s ability to transform the organization. He also told me that Scrummasters are responsible for team health. Initially I liked this thought. It is great to know I have an expert holding my hand and coaching me along the way as I go down the Scrum path to an Agile process.

Over time I have started disliking the thought of one person with so much responsibility. In my experience there has been shared ownership across the leads and managers when we migrated to Agile. There were definite Agile experts on my teams, and we frequently asked those experts for guidance, but we never asked the experts to own the process or team health. We did this collaboratively as a leadership team.

I have found this method successful because we do get expert opinion, but we do not relinquish ownership of the process to one person. In my experience this co-ownership leads to an Agile process that is better adhered to because it was created together. Now that we have outlined a process for obtaining management support, let’s discuss a process for getting the team to buy into Agile.

The project team

An Agile team is different from the average development group. Team members come across as poised and ready for where ever the project may lead them. An Agile team member does not fear uncertainty. They look forward to the challenge and they know they will succeed.

Where does this air of self-assurance come from? Is the attitude reflective of the type of people that were hired? Or is it reflective of the processes that are being used? Is the attitude a byproduct of executive support? Does the confidence come from a history of successful deliveries?

The answer to all of the questions above is yes. Each of the items described above supports the effectiveness and self-reliance that is inherent in an Agile team. In some ways creating an Agile team is like baking a cake. You can obtain the ingredients exactly as the recipe requests. You can bake at the suggested temperature. You can let the cake cool the specified time before applying the icing. But what happens if you are at high altitude and you forget to make the necessary adjustments? The cake rises too quickly and then turns out too dry. Or what if someone jumps up and down in the kitchen while the cake is baking? The cake collapses and never rises.

Next I will give you the ingredients for creating your Agile team.

Culture and roles

I find it hard to describe Agile team culture in a sentence, but I can easily describe it with several words. The words that come to mind are: collaborative, open, passionate, courageous, honest, light-hearted, driven, synchronized, customer focused, funny, responsible, innovative, and successful.

The culture is one of low politics and high transparency. Words are honest but not abrasive. Status is discussed in matter-of-fact terms. The team focuses on the situation, not the person. Estimates are honest. There is no padding to make the work easier to do. There is no lying about how long it will take to appease management.

Another nuance of an Agile environment is the roles that team members play. Except for the use of Scrum, Agile does not specify what team member roles should be. In my experience this has not been an issue. The teams I have worked with did not change their roles after they migrated to Agile. We still had developers, testers, project managers, product managers, customers, DBAs, and Operations.

What did change for those teams was attitude. After we migrated to Agile I rarely heard a team member saying something like “development is not responsible for that” or “quality determines when the code is acceptable”. I saw many more team decisions and I saw much more collaboration around problem solving. A problem was not tied to an area, and they had to solve it. It was tied to the project and the team had to solve it. The team focuses on the goal, not their job description.

The last item related to culture is diversity. If you do not have a diverse team your Agile process can lead to groupthink. Groupthink happens when a team wants to get along with each other so desperately they will not voice their opinion when they disagree with an idea. This is a definite danger with Agile. People assume collaboration means harmony and always getting along. They think if they start agreeing with each other all of the time they are being collaborative.

A classic groupthink example is the space shuttle disaster on January 28th, 1986. The space shuttle Challenger was preparing to launch on a cold day, colder than any other space shuttle had been launched on. One of the engineers from a company that supplied parts to the space shuttle warned that there could be risk in launching. He was concerned that the O-ring seals his company provided may fail in the low temperatures since they had never been tested below 53 degrees Fahrenheit. The engineer shared this concern during a teleconference with NASA and NASA urged him to reconsider his recommendation to not launch. The pressure from NASA persuaded the company to acquiesce to the request and overrule their engineer’s warning. Subsequently the O-rings failed just after launch, leading to the death of the entire Challenger crew (Griffin 1997).

The reciprocal of groupthink is diverse opinion that is spoken freely. This is what you want in your Agile environment. A good example of this is occurred during the Apollo 13 space mission. In this instance there was an explosion aboard the spaceship on its way to the moon. The ship and crew were salvageable with a little luck and some spectacular collaboration.

As the crew experienced various issues in trying to return to earth, the support team on the ground went through days of brainstorming and collaborating to solve the problems. No one team member had more influence than another in suggesting a solution, and “getting along” was not a requirement. As problems were discovered ideas were discussed passionately until the group reached consensus.

Culture is not an optional ingredient in your Agile recipe. The majority of the team must embrace the Agile culture else you will not be Agile. You will just be a team that calls your self Agile and you will go about business as before.

Let’s take a moment to look at the building block of the team, the individual.

Characteristics that influence individual performance

Everyone on your team does not need to be competent and mature, but you want to put a system in place that breeds competency and helps the entire team get there over time.

Just like in traditional development, competency alone does not guarantee team success. There are several factors that affect the productivity of an individual. Let’s review a few of them.

Motivation and reward structure

A talented, mature individual will not stick around to work on your Agile projects if his efforts are not rewarded. A person who is talented can frequently choose where they want to work. It is up to the company to create an environment that attracts and retains talented individuals.

In simplest terms, behavior reflects incentives. What are the incentives you will provide to attract talented individuals to your Agile team?

Consider the following items related to motivating and rewarding the individual:

Is the mission of your company clear? Has it been clearly communicated to each individual? Employees will want to know where the company is going and how their projects tie to the vision.

How is health of your company? Are you doing well financially? Are you a start-up fighting to survive? Company health can tie to motivation on in two ways. First, if you are healthy and growing, you can convey this message to employees and tell them that there is stability, growth opportunity, raises, and potentially equity. If you are struggling to survive the message is the importance of their project and how it affects the destiny of the company. Everyone wants to work on projects that are important.

The Agile environment is going to stress the value of the employee beyond their job title. They will make management decisions and they will be responsible for proactive communication. Talented individuals will welcome this environment. Employee evaluations should recognize and evaluate collaboration skills.

Career stage

As you migrate to Agile you will need to consider various approaches to migrating your employees to an Agile mindset. To help you determine the approach, consider where each employee is in regards to their career. Here are the main stages and suggested approaches.

The New employee: These employees are in a stage of rapid learning and trying to understand the company and processes around them. They are dependent on others to get things done, and they are working to become independent from support. These employees will enjoy learning Agile because it will level the playing field for them. They will be at ground zero, just like senior employees, and they will be comforted by the fact that everyone is learning Agile together. They should also do very well using the methodology because they do not have a lot of previous experience to bias them.

You do not have to do anything special with these folks. Just ensure they get the same training as everyone else, and that they are offered the same opportunities as other team members.

The Individual contributor: These employees will make up the bulk of your teams. They are not new and they are not supervisors or managers. They have a medium to large amount of experience, and they may have chosen to not become managers, but to become a guru in their area.

These folks require the most management and you need to address their needs individually mentor. Some general tips for motivating these employees are:

- Give them an area to own and be responsible for in your migration.
- Give them an opportunity to use and share their expertise.
- Give them a chance to be innovative and unique.

A lot of these folks are looking for growth and will embrace Agile. Some of these folks will just be getting comfortable with the way things have always been done and they will resent having to learn another new thing. Be patient with the “resenters” and remember them when the time comes criticize the Agile design. Their feedback will be valuable.

The Coach: Employees at this stage are motivated by sharing their experience and mentoring others. They are also looking for an opportunity to renew and revitalize them self. An Agile migration project is just what the doctor ordered for these employees. Give these folks leadership opportunities during the migration, such as resolving design issues or leading the team to consensus. They can also be on the forefront of receiving Agile training and they can mentor novice employees on the process.

Now that we have a process for getting management and the team on board, let's discuss the most important foundational piece, executive support.

Obtaining executive support

Ralph Waldo Emerson said “Nothing great was ever achieved without enthusiasm”. For those of you with significant business experience, you know that “Nothing great was ever achieved without executive support”. This is not true because of executive team impact; rather, it is true because executive teams will stop anything they have not endorsed. They will want details, justification, meetings and more meetings if they are caught by surprise on a major initiative they have not endorsed. A migration to Agile would be considered a major initiative in most companies.

If you surprise the executive team that may allow you to still go forward but energy and momentum will be lost if you get sidetracked by not involving them at the start.

In this section we will show you how to obtain executive support by addressing the specific needs of the group.

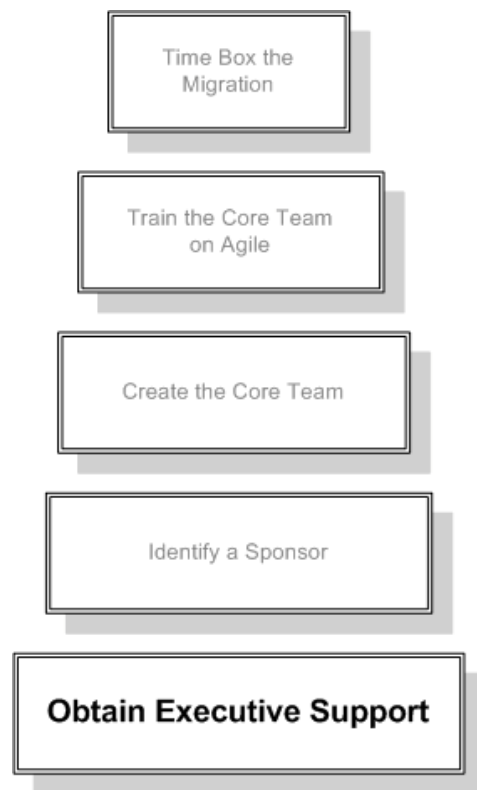


Figure 3. The five key steps in preparing for your Agile migration. Executive support will provide a foundation for the migration and clear roadblocks for the team along the way.

A few things are guaranteed when you meet with your executive team. They will want answers to the following questions:

- Why you are pursuing this initiative?
- What is the value?
- What are the costs?
- What are the risks?
- What will it do for me?

Let's look at some potential answers to these questions and determine which ones best fit your situation.

Why pursue agile?

There are a variety of reasons why you are pursuing Agile and what the value is. Here are the ones that resonate with executives:

- There is no methodology. You really do not have any processes or framework in place and you do projects different every time with varying results. You are pursuing consistent, successful delivery of projects.

- Your current methodology is struggling to keep up with the volume and volatility of your work. You are looking for a way to deal with projects that need quick turn around, but have minimum requirements definition.
- Your customer is not happy. The customer feels disconnected from the process and they feel their needs are not being met. You are looking for a way to get the customer more involved in the development process and improve their satisfaction. (Important to note here - If you involve the customer more in the process their satisfaction rating will improve even if your deliveries do not. They will have more empathy for what it takes to create value for them and in turn they will have a more positive feel about your company and the development group. In this instance there is truth in the saying that perception equals reality.)

The items above are solid reasons for migrating to Agile. However, your executives may be concerned that the migration is inspired by something else such as boredom or trying to become cutting edge. Perhaps team members are looking for a good resume bullet. There is nothing wrong with migrating to Agile for a resume bullet or to modernize your processes, but these need to be secondary benefits. Migrating to Agile is not free and it should only be pursued if it benefits the company.

One last note on value. This book includes many statistics related to the value of Agile. These statistics relate to business satisfaction after migration, cost reduction, and improved quality. These statistics are good for appeasing the Agile detractors in your company and the employees who try to measure everything in terms of probabilities. You can use these statistics to justify your migration to Agile and, for lack of better words; your backside will be covered if the migration goes awry.

But what statistic will show you how much effort a company put into their migration? What statistic will show you how passionate the employees were that brought Agile into these companies? Where is the statistic that measures how much executive support was obtained before the migration started? How much training did employees receive? In effect, the statistics only prove that Agile is not a fad because enough people have used it to create statistics for it.

There are so many intangibles in a migration to Agile that I would feel guilty if I recommended it based solely on statistics. I recommend Agile because I have seen it work in several environments. I recommend Agile because I have enough experience to know what the common issues are related to software development. I know Agile principles address these issues and ensure my probability of successfully delivering projects.

The cost of migrating

What will you say when the executives ask about cost? In the model proposed your main expense will be having a knowledgeable Agile person or company come in to train the team. This is usually 2 to 10 days of training, with a several phone calls and one-off consulting sessions post training. A ballpark number for the consulting/training assistance is \$5,000 - \$10,000.

The other expenses are less tangible. They are frequently labeled as “soft” expenses because they do not add to company expenses but reallocate existing resources. This will be true of the Agile Core Team. Over a 3 month period the Agile Core Team may spend 10% of their time working on the new Agile methodology. Other costs are relatively minor, such as printing out materials to support training.

The last “expense” of note is slower delivery. You can expect the first few projects to be slower as the team gets comfortable with the new processes and each other. After an acclimation period the team will gel around the process and you will deliver high priority features sooner than before, but patience is necessary with the first few projects.

An analogy that comes to mind is auto reviews. I frequently read car magazines and I enjoy the road test reviews on new vehicles. Almost every review will lament the position of the shifter, the strange angle of the seats, or the lack of cup holders.

I will buy the same magazine 6 months later and it will contain the long term road test results for the same vehicle. Frequently the extended review will say “Although initially quirky, the position of the shifter becomes intuitive with long term use and simplifies the shifting process. We also found the seating position to be excellent for long distance road trips.” Your migration to Agile will be similar. Once you get comfortable with how it works you will find it “becomes intuitive with long term use and simplifies the development process.”

While discussing the cost of migrating to Agile we should also consider the cost of not migrating. If you reflect on the items listed in section 4.1.1, the reasons for why you are pursuing Agile, what happens if you do not address those issues? The consequences can include:

- Declining customer satisfaction
- Loss of key employees
- Missed deadlines for compliance related projects
- Lost sales
- Lost opportunity

Whether you like Dr. Phil or not, there is a question that he frequently asks his guests that relates to migrating to Agile. The question is “How is your current process working for you?” This is Dr. Phil’s subtle way of saying what you are doing is not working, period. You need to make a change.

The risks in migrating

The next question is “what are the risks?” If done correctly I believe the risks are minimal, but I will list some that can occur with poor management of the migration process:

- You can fail on a critical project if you pilot your Agile methodology on it. The first few Agile projects should not be mission critical. You need to start and test on projects with medium priority and work your way up to critical ones.
- The migration can fail if it is executive driven and there is disregard for pursuing employee buy-in.
- There can be impact to projects if employees hear about the work the core team is doing and decide to experiment without guidance. I have seen this happen, where a team will take one Agile practice and try it with disregard for how it needs to dovetail into the upstream and downstream processes.
- With improper training the methodology can foster cowboy coding and insufficient documentation.

Rewards for the executives

The last question is “What will the Agile migration do for me?” There is nothing wrong with this question. We all have career needs and no one likes to undertake a venture that puts their career at risk. It is fair to ask what Agile will do for me on a personal level.

The answer to this question is usually unique. More than likely the executives will never tell you the answer to this question directly. You will have to deduce the best way to make them look good. Here are a few ways I have seen an Agile migration make executives look good.

- Agile allows executives to acquire new skills and knowledge which will increase their value to the company and increase their chances for promotion.
- The move to Agile provides an opportunity to demonstrate leadership skills by leading a major organizational change. The executive sponsor will reap this reward.
- The migration to Agile will lead to more wealth. Like most of us, executives care about their compensation. Migrating to Agile will lower costs and increase revenues, which should also lead to an increase in stock value, or if you are a small company, survival.
- Fewer people issues. All managers dislike dealing with people issues. The Agile work environment is more satisfying and the executives will find themselves dealing with fewer employee issues. They will also be pleased to see employee retention rates increase.
- Fewer customer issues. As mentioned earlier, customer satisfaction increases with Agile. A happier customer leads to more pleasant discussions with the executives.

Communicate frequently

You need to communicate frequently with the executive team and keep them abreast of the progress the core team is making. Although it may sound a bit anti-Agile, you may want to publish a weekly status report that provides an overview of progress made, status related to projected schedule, risks being managed, and issues encountered.

You should also schedule a recurring meeting with the executive team to interact with them face to face. The meeting will allow you to add more depth to status and continue with the Agile education of the executives. I believe you should meet with the executives about every two weeks. This timeframe works well with their busy schedules and also allows you to spend more time managing the migration and less time reporting on it.

You should also encourage informal interaction with the executives. Some executives may like a one on one session with the core team. Welcome them with open arms.

You may also have executives who like to drop in during core team meetings to listen in on activities. This is a positive too, just make sure the presence of an executive does not intimidate the team and they can stay on course with the task at hand.

All of the items above are great ways to communicate with the executives, but the best way is to have someone within their ranks directly tied to the migration. You need an executive sponsor.

The role of the sponsor

Your executive sponsor will be the liaison to the executive team and help clear hurdles for the team as they develop the new methodology. The most logical path is to find the executive most closely related to software development. Frequently this is the VP of Development, or potentially the VP of Product Management.

If your desire to migrate to Agile is being driven from the “doers”, that is not at the executive management level, it is best to work your way up the ladder to obtain executive support. For example, if you are the project manager and you are proposing the change, you could discuss it with the director of software development. If he buys in, you could ask him to help you take it to his superior, perhaps the CIO. You could do a dual presentation to convince the CIO of the need. Once you obtain your sponsor should you expect them to play three major roles.

In their first role they will keep the executive team up to speed on the migration outside of the scheduled status meetings, and they will act as a champion for the migration. They will help your team acquire funding for the migration and remove roadblocks at the executive level.

In their second role they will represent the organization, ensuring that the Agile migration project is in line with the organizations goals and strategic objectives. They will help the Agile team ensure success and minimize risk to protect the organizations investment in the change.

In their third role they will provide leadership for managing the organizational change that needs to occur with a shift to Agile. This would include working with the executive team to create a rewards structure that encourages Agile behaviors.

These three roles can manifest themselves in many ways. Here are some typical activities of a project sponsor.

- Help the team define success for the migration.
- Help the team obtain outside help when needed.
- Ensure that the new methodology works within the organization culture.
- Help with migration team morale and recognize successes along the way.

Lastly, help your executive sponsor if they do not have a technical background. Train them on how software development works and the intricacies of Agile. Be patient if they need time to digest how it all works.

Now that we have a feel for the cultural needs of the executives, let’s take a moment to discuss working managers.

The role of the core team—ensuring company buy-in

The key to a successful Agile migration is having the change driven from within. The change needs to be driven by key players throughout the company. Once this team is created they will be evangelists to the entire company.

The role of this group, which I call the Agile Core Team, is to learn as much as they can about Agile and use this knowledge to outline a custom Agile methodology for the company. The team will collaborate and reach consensus on new processes, then mentor project teams as they use the Agile techniques.

This core team is powerful and influential for three reasons:

- They are not a part of line management. There will be very few members from the management ranks but the majority of the team will be “doers”. The people that actually design, build, create, and test the code. This will add to the credibility as the methodology is rolled out to the company. It is not a management initiative being forced upon everyone; it is coming from real people who will be a part of the project teams.
- Since the team is composed of doers they actually know the ins and outs of developing in your environment. This is different than when consultants come in suggesting standard practices and disregarding the realities of a specific company. The Agile Core Team has experience with your company and they will use that experience to develop a methodology that knows what to keep and what to discard within the existing practices.
- Remember our earlier discussion of awareness, buy-in, and ownership? What better way to create awareness than to have Agile Core Team members come from each functional area. Imagine a member being from quality and going back to the quality team and telling them what is going on with the new methodology, or a developer doing the same with the development team. Having team members from all areas will initialize awareness across the company.

Many companies use outside consulting to get their methodology going. I have seen several companies choose to go with Agile methods such as Scrum, and then have a 3rd party come in and train, design, and deploy the methodology. In my opinion this approach is not as effective as growing the methodology from within. Creating it from within the organization addresses all of the issues with ownership. It is hard to get a team to buy into a process that was forced upon them. Note that there are occasions when an organization is so dysfunctional that it needs to have a methodology forced upon it. This is the exception, not the norm.

Obtaining team members from all areas

Once you obtain executive support you can pursue creation of the Agile Core Team. Your sponsor will probably suggest managers for the team, but you need to remind him that part of the power and influence of the core team is they are “doers”. You might also find yourself pursuing the best and brightest people from each area. People with a positive attitude and a pro-Agile mentality. People that are open minded to change. These would be excellent attributes to list on a job opening, but would they be reflective of your current employee mix, the people that you want to embrace the new methodology? Probably not.

If your company is like most you probably have some mix of the following:

- Brilliant and collaborative people
- People that are brilliant but difficult to work with
- People who challenge every initiative
- People who loathe change and avoid it at all costs

You need to make sure the makeup of the core team is similar to the makeup of the company. This will help you obtain buy-in from all types when you begin roll-out.

After determining types of folks for the team, you need to determine team size. A group large enough to capture a diverse set of perspectives but small enough to be, yes, “Agile”. I suggest a number somewhere between 5 and 10 people. Note that if the team is larger you can still make progress when a team member is pulled for a production issue or is out due to vacation or illness.

To give you a feel for creating your own team, let's return to Acme Media. We find that Wendy Johnson has obtained the CIO, Steve Winters, as the executive sponsor for the Agile migration. Amazingly Steve has asked to be on the core team and he says he can participate in the 6 hours per week requested of team members.

Wendy and Steve have also identified their Agile Core Team and they have received approval from the managers of the people selected. Wendy and Steve worked hard to get a diverse group of people on the team to allow many perspectives to be considered in creation of the methodology. Their team member list is can be seen in table 4.1. You will notice that members are from various functional areas and they all have different points of view on what a methodology should do, just like your team will.

Table 1 Acme Media's Core Team. Core teams are composed of cross functional team members with various levels of Agile knowledge. The diversity of the team works well for scrutinizing the new process.

Functional Area/Role	Name	Background
Sponsor/CIO	Steve Winters	Six Sigma enthusiast. Does not believe in change for the sake of change. Willing to pilot Agile and see if the benefits manifest.
Project Management	Wendy Johnson	Frustrated with the status quo. Lately quoting Dr. Phil "If it ain't working you got to try something else". Wants a methodology that reflects the reality of how software is developed.
Development	Roy Williams	Familiar with XP development techniques, but very comfortable with the waterfall process that Acme has used the last few years.
Quality Assurance	Vijay Kumar	Concerned that Agile will bypass or minimize the need for testing. Experience working in an ISO environment. Frequently says "document what you do, do what you document".
Operations	Matt Shiler	Lives in a stressful world of managing production issues and deployment of new functionality. Worried that he will not have enough time to work with the core team.
Requirements	Wes Hunter	An Agile zealot. Been looking forward to this day for a long time. Dedicated to making Agile work at Acme. Works with Product Management to refine feature design for customers.
Architecture	Keith Gastaneau	Wants to make sure that an Agile methodology does not bypass good architecture practices, and there is enough time to build the infrastructure needed for projects.
Product Management	Peggy Romani	Unfamiliar with Agile but excited of the promise to embrace the customer and changing requirements. Identifies target markets and strategic needs of the products.

Just like Acme, you will need to get manager approval for the employees you select for your team. The managers will probably be looking for a time estimate from you. The first few weeks I like to see the core team meet twice a week for 2 hours each meeting. You can also assume each team member will get a couple of hours of Agile homework a week (researching existing development methods, etc.) A good number to give the managers is 6 hours a week for 3 months. The duration will decrease over the 3 months. You may see the weekly meetings reduced to one hour, but to be safe still ask for 6 hours per week for 3 months. Better to promise late and deliver early.

After team selection you need to meet with the members and set expectations.

First meeting of the core team

Once your team has been named you should schedule a kickoff meeting to set expectations and goals. Similar to meeting with your executive sponsor, you will need to start the meeting by telling the team why the company is migrating to Agile. The verbiage will be slightly different with the core team, with less focus on financials and a more focus on process.

The kickoff meeting

To see an example, let's look at the presentation Steve Winters is using at Acme's kickoff meeting. Steve starts the meeting with the following bullets:

- Acme Media's web division was no longer a supplemental site to the television station. The web sites had their own audiences and advertisers.
- With the increase in popularity of the web sites, the backlog of new features and application requests has increased by 70%.
- Many of the feature requests are time sensitive. If the requests cannot be completed soon our competitive advantage will be lost.
- Our existing development processes, where we have them, are not working well with our tight deadlines or with the evolving requirements.
- We need to research a better way of dealing with urgent projects.

As you can see, Steve's message was tailored more to the project team than it was to an executive group. He spoke indirectly to revenue by saying "lost advantage" and he mainly targeted process improvement. The best thing Steve said was "the web sites are no longer supplemental sites to the TV station" and "the web sites have increased in popularity." Steve was telling the team that their work was important.

You should follow Steve's example during your migration, especially emphasizing the importance of the work the team does and how valuable the methodology they develop can be.

Tough questions

Of course everything will not be roses at your kickoff. You can expect difficult questions and perhaps "attitude" from some of the core team members. Here are a few of the questions and comments you are likely to hear during your kickoff:

- We cannot create the methodology. We don't know anything about Agile.
- We need consultants to do this for us.

- We have tried to change before and it failed.
- What is our role?
- What is the role of the executive sponsor?

The answer to the first question is easy. The team will get trained and soon they will have a good working knowledge of Agile. If you are lucky, a few team members are already versed on Agile to help bring the team knowledge level up.

On the second question they are half right. You will bring in a consultant or Agile guru to train the team on the fundamentals, and perhaps discuss what other companies have done with their methodologies. But the consultant will not create the methodology for them. They will do that and later they will be glad they did.

The third question is a warning sign to you if you do not know the details of a past failure. Was it due to a methodology being forced on the team? Was it due to waning executive support for the change? Do your homework if you learn of a past failure and make sure your plan covers the lessons learned from previous ventures.

Assuming the issues related to a previous failure do not exist anymore, you can explain to the core team why the migration should be a success this time:

- The design will be created by experts who know the business well - them.
- They will not be forced to remove a legacy process if it is proven and adds value. If this is true there is a good chance it is already an Agile process.
- The approach will not be shotgun. The methodology will be built iteratively and it will be deployed iteratively to mitigate risk. In addition it will not be beta tested on a mission critical process.

Answering the question about their role is simple and clear. The team will learn about Agile and use this information to create the Agile methodology. In quick summary they will:

- Train
- Document the existing project and development processes.
- Determine what to keep and what to discard with the current processes.
- Compare the existing process to a pure Agile one.
- Design a new methodology based on Agile principles and the reality of the work performed.
- Get feedback on the design and tweak it.
- Take the design for a test run on a low to medium level priority project.
- Learn from the test run.
- Continue refining and testing until the methodology is solid enough to be used on all projects and the team is comfortable with the processes being used.

As mentioned earlier, the role of the executive sponsor is to clear roadblocks for the team and to be the liaison to the executive team at large.

Your role in the migration

Another question not listed but potentially asked is what is your role? Assuming you, the reader, are the leader of the core team, there will probably be questions about your role. If you are a manager you want to make it clear that you are all equals during team meetings. Titles and status will be left at the door including your own. As leader you will help organize the meetings and report status to the sponsor. You will also act as facilitator and help the team reach consensus on design ideas.

Training the core team

Training needs to happen within a few days of the kickoff with the core team. Determining the level of training is tricky. You want to provide enough information so the team understands the Agile principles and their value. You do not want to train to a point where you have handed them a methodology - especially somebody else's. You want them to combine Agile principles with their knowledge of your business to create a methodology that is effective for your company.

You will need to use your own judgment on how deep to train depending on how well the principles are being digested and how creative you perceive the team to be. Here is a suggested outline for training

1. Explain to the team where Agile came from, what makes it works, places where it is working, and why it has not faded away. This training should take 4 to 8 hours.
2. Give the team a few days to absorb the principles then train them on the phases of Agile. We have chosen phase names that map well to names used in traditional software development, which helps with the training process.
3. Once training is complete the team will begin the design process by reverse engineering the existing processes.

Keep the energy flowing—goals and milestones

As mentioned earlier, you need to have regularly scheduled meetings of the core team to maintain momentum. You also need to create a list of milestones for the team and time-box their work. This is critical because a team can easily spend a year working on the creation of a methodology. Just like Agile development, time-boxing the design process will force decisions and allow for early demonstration of the methodology to validate it. The methodology will be a living thing that is refined continually. The first priority is to bring it to life quickly.

To get a feel for time-boxing the redesign, let's look at the plan that Acme Corporation has outlined. In this example the total duration is two months to reach the point of being able to test the methodology on a project:

- Train the team on Agile principles, phases and optionally with a real world example. – 1 week
- Document the existing development process. – 2 weeks
- Review the existing process for deficiencies and identify areas to change to make them more Agile – 1 week
- Outline a new Agile process with the proposed changes – 2 weeks
- Get feedback from non-team members on the proposed design – 1 week

- Refine the design based on the feedback. – 1 week
- Pick a test project to try the new methodology on. – 1 week

You should estimate your design work to take anywhere from four weeks to eight weeks, depending on core team availability, business model complexity, and the dynamics of the team. Some teams reach decisions quickly, some will debate for a while before reaching consensus. Use your design milestones to push the team to decisions if they get stuck along the way. Remind them that the design is not permanent and will be refined as you all learn.

Developing a communication plan

In the process we have outlined so far, the core team knows what is going on and their managers have a high level idea of what their employees are doing. Outside of these people, the Agile migration project has not been communicated to all of the employees of the company.

A question that Acme Corporation has to answer, and you do too, is how to communicate the migration to the development group at large. There are two approaches that can be taken depending on how you categorize your company.

A category one company is progressive and somewhat cutting edge. A good portion of the employees may have already heard about Agile and they are curious about it. The culture embraces change and they take pride in their open mindedness. The culture probably embraces innovation. Some companies that come to mind are Apple, Google, and Yahoo.

If your company falls into this category you can communicate freely on the status of the migration, including issues encountered and adjustments you are making along the way. You may want to post a weekly update to your intranet or email one to the development group at large as you progress.

A category two company has limited knowledge of Agile. The culture may struggle with change and there could be historical issues in trusting executive management. These companies may have limited success in being innovative.

If your company has any of these characteristics, I suggest the following ideas for your communication plan and method:

- Get the executive team on board as soon as possible. Keep them up to speed on the progress and findings as you progress. This group will help you when you officially announce the methodology to everyone.
- Outside of the executive group, maintain a low profile as the core team is developing the methodology.
- Allow the core team to communicate to their peers and their team informally about their work. Someone once said the best way to spread information is to label it as a secret. Don't label the migration as a secret, but don't spend too much time communicating about it before you know what it is. The informal communication will also kick-off the awareness phase.

There is probably fear of the word "Agile" in your environment, so do not use the word when communicating with others. You should also consider labeling your methodology in a way that has no connections to Agile. Your goal is to get the company to adopt your new methodology. You do not care what they call it. So consider a name like ACDC, Acme Corporation Development Cycle versus ADLC, the Agile Development Lifecycle.

Most Agile migrations will have issues with skeptics and people who are resistive to change. As you get closer to testing your methodology you need to embrace these people and give them the floor with the core team. Let them discuss their concerns and critique the proposed design. The value in this is twofold. First, a good deal of the time they will identify a weakness in your process. Second, they will be converted to advocates if you listen to them and you are able to make some of the changes they suggest.

Summary

Establishing an Agile culture is just as important as establishing an Agile process. An Agile culture must be supported by management or the team will struggle to succeed. Train your management team on Agile concepts and establish their support before asking your team to.

Train your managers on the soft skills that support an Agile environment such as listening and team building skills. Teach your managers to shepherd the team and to teach the team how to manage themselves.

An Agile team is successful because it is honest and project status is transparent. Reward your team for demonstrating these values.

Assess your project team member needs and give them a migration role that supports their unique needs.

Kick start your migration by establishing support at the executive level of your company. Be ready to quantify the value of the migration to the executive group.

Create a core team that is composed of employees throughout the development process. This group will tailor an Agile process to your environment and drive the migration to Agile from within. Make sure your core team is diverse and represents many skill sets and perspectives. Make sure the core team is clear on the value of Agile before asking them to engage in the migration process.

Establishing your first pass at an Agile process can become a run away train if you do not time box the effort. Limit the time you spend creating your lifecycle and remember that you can, and will, improve it over time.

Once you have initialized your culture and established a core team you will be ready to create your custom lifecycle.

Independent Report by Forrester Research, Courtesy of MKS:
"Selecting the Right Requirements Management Tool - Or Maybe None Whatsoever". Forrester Research recently evaluated the requirements management tools landscape across four criteria: Baselineing, Linking & Tracing, MS Word Support, and Workflow. MKS is the only vendor in the evaluation to fully satisfy all of the critical features in Forrester's evaluation. Download your copy of the independent report, courtesy of MKS:

<http://www.mks.com/mtspring-rmreport>

Are you looking for the right development tools for your task. A new directory of tools related to software development has been launched. It covers all software development activities: programming (java, .net, php, xml, etc), testing, configuration management, databases, project management, modelling, etc.

<http://www.softdevtools.com/>

The Software Quality Assurance Zone is a repository for resources concerning software testing (unit testing, functional testing, regression testing, load testing), code review and inspection, bug and defect tracking, continuous integration.

<http://www.sqazone.net/>

Advertising for a new Web development tool? Looking to recruit software developers? Promoting a conference or a book? Organizing software development training? This classified section is waiting for you at the price of US \$ 30 each line. Reach more than 50'000 web-savvy software developers and project managers worldwide with a classified advertisement in Methods & Tools. Without counting the 1000s that download the issue each month without being registered and the 40'000 visitors/month of our web sites! To advertise in this section or to place a page ad simply <http://www.methodsandtools.com/advertise.php>

<p>METHODS & TOOLS is published by Martinig & Associates, Rue des Marronniers 25, CH-1800 Vevey, Switzerland Tel. +41 21 922 13 00 Fax +41 21 921 23 53 www.martinig.ch Editor: Franco Martinig ISSN 1661-402X Free subscription on : http://www.methodsandtools.com/forms/submt.php The content of this publication cannot be reproduced without prior written consent of the publisher Copyright © 2008, Martinig & Associates</p>
--
