METHODS & TOOLS

Global knowledge source for software development professionals **ISSN 1661-402X**

Spring 2006 (Volume 14 - number 1) www.methodsandtools.com

Do We Have a Standard or Lack of Innovation?

The globalisation of the economic world is a trend that has certainly been sustained by the development of technology these recent years. Internet makes the diffusion of knowledge and ideas easier between the parts of the world that are connected to the web. We have also seen a decrease in the diversity of available ideas and products. At the end of the last century, the choice was wider in the computer industry. US companies like Digital, Data General or Wang were competing in the world of minicomputing. European countries had also their national computer industry with companies like ICL, Siemens, Bull, Norsk Data or Olivetti, each with its own operating system. Now the choice of a supplier is reduced and there are limited alternatives besides Unix/Linux and Windows in the operating systems. The same evolution happened in the field of software development methods. In the 80s and 90s, the SEI was establishing the basis of the CMM; independent thinkers like Yourdon, Jackson or Martin were promoting their approaches; an EuroMethod project was funded by the European Commission to consolidate national methologies like SSADM or Merise. Then the rise of object orientation provided a new breed of modelling techniques that included the object paradigm. But in 1994 begin the creation of the UML by merging the Booch, OMT and OOSE approaches. UML became an OMG standard at the end of 1997 and since then we have had few diversity on the modelling front

There are many reasons for this. Even if some modelling tools continued to implement non-UML techniques, there has been a decline of "national" editors that were the champions of their homeland methods. Some have survived, either due to a large national market or by looking for other niche markets like process modelling, but they lack the resource and motivation to evolve and promote their original approach. People have also realised that modelling techniques were not the silver bullet to solve the problem of delivering applications that meet end users' needs. Finally the UML/OMG standardisation process allows to involve different participants. Many modelling researchers should have thought: "if you can't beat them, join them". Even if a standard like UML improved the overall level and the acceptance of modelling techniques, it is damaging that a collateral effect could be to limit the amount of new thinking.

[illegible]

Inside

Use Cases and Implementing Application Lifecycle Management Systems

Paul Bowden
MKS Systems, http://www.mks.com

Abstract

The UML technique of Use Cases provides a powerful tool to elicit and document stakeholder requirements. While usually used in a traditional development context, use cases can be deployed with good effect when implementing an enterprise scale Application Lifecycle Management (ALM) system.

Application Lifecycle Management encompasses the processes and tools employed at all stages of development: from project inception and requirements gathering to deployment and maintenance. With this much broader scope we must take care when planning and deploying several complex interacting systems. This article describes how use cases can help us plan and implement ALM processes and tools.

Introduction

Implementing an enterprise application lifecycle management system is very different from giving your developers a new version control system to put their source code in. Enterprise ALM affects processes that control the whole software development lifecycle and very likely other areas of the business. For instance an application that supports the financial management of the business will most likely be subject to corporate governance regulations and our systems must be able to demonstrate compliance. Enterprise ALM is always driven by the goal of delivering value to the business and alignment with business strategy, so we should be clear about our scope, the business stakeholders we are going to impact and how we can measure our success.

As with all activities, planning is key. What tools and techniques can we apply to reduce risk and ensure a timely implementation that delivers value? This paper will focus on an important technique in UML, use cases, which can help when eliciting requirements, documenting and planning the implementation and validating we have the systems we need.

Use Cases in an ALM context

A use case is a sequence of actions a person takes to achieve a specific goal. In more detail we can say:

We identify a set of activities, performed in a certain order

We know who performs the actions. The “doer” is usually referred to as an actor. This acknowledges the fact that we may be interested in a class of people, a particular role or indeed another system.

- There is a defined result to the actions: a goal.
- The use case has a descriptive name.
- The use case is usually summarised in a diagram linking the actors to the use cases they perform.

Use cases being descriptive text within a more or less formal framework, we can develop, discuss and manage them with any of our stakeholders.

By way of a very simple example, thinking in terms of ALM, we can quickly come up with: **Check out file**. Our actor will be something like **Developer**, and we can describe the use case thus.

Check out file

Actor: Developer

The developer selects one or more files from the repository and selects Check-out.

The files are marked as locked by the developer and are write-enabled in the personal workspace.

The second paragraph in our use case identifies the "post-conditions", that is, the state of the system after the use case has been completed. Where important, we can also specify pre-conditions for the use case. Pre-conditions specify things that must be true before the actor can execute the use case; for example they have logged on.

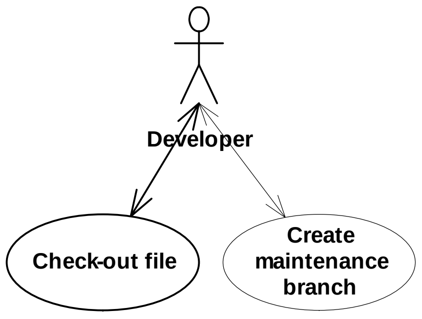
The use case can also explore alternate sequences of action or “paths”. The non-exceptional path through the use case is sometimes called the *happy path*. It will often be instructive to explore *alternate paths* through the use case. The Check Out use case above provides a good illustration. What happens if the actor tries to check out the file and another developer already has it locked?

Alternate path

Another user has a lock on the revision that the actor is trying to checkout. The actor branches the file and locks the branched revision. The actor must later merge their changes into the modified file.

This use case expresses an important policy we will implement: developers won’t wait for the lock to be released or make changes locally; they will branch around the lock. Of course, we could choose another policy but the path through the use case has forced us to make the decision explicit.

The use cases are usually illustrated with a simple diagram showing the actors and the use cases they execute.



The diagram complements the detailed text of the use cases by giving a visual representation of the use cases we are considering. This can be useful in workshops where we want to focus discussion on a particular area of the implementation. For instance, how and when we will branch the codebase to support post-release maintenance.

Templates

There are many templates for use cases available as part of methodologies or from various web sites documenting best practice or as part of paid-for consulting. The precise form use cases take is a hotly debated topic and the best advice offered seems to be to do what works for you. However, since we are likely involving a broad range of stakeholders our guiding principles should be clarity to a wide non-technical audience.

Use cases and requirements

Use cases capture functional requirements but situations may a rise where the development of requirements is separate from that of the use cases. For instance requirements may be prepared in advance by the (internal or external) customer. In this case the requirements map to use cases. This mapping will form a traceability matrix where all requirements are fulfilled by the use cases allowing us to demonstrate that we have satisfied the requirements.

Use cases and functional specifications

Use cases and functional specifications are complementary and can be used together if the need arises. It is obviously important that they are consistent. A functional specification can capture much more detailed information (for instance all the field definitions in a workflow) that would simply cloud a use case and negate its usefulness.

Planning an Implementation

If, when embarking on an enterprise implementation, we try to identify all the use cases that the developers, architects, managers, etc. will execute we'll have an enormous job on our hands. To avoid so called "analysis paralysis" we need to focus our attention on analysis activities that deliver value. Examining routine check-ins will not be a good use of time. However, a detailed description of how a check-in triggers an automatic integration build will be. This use case is worth considering because

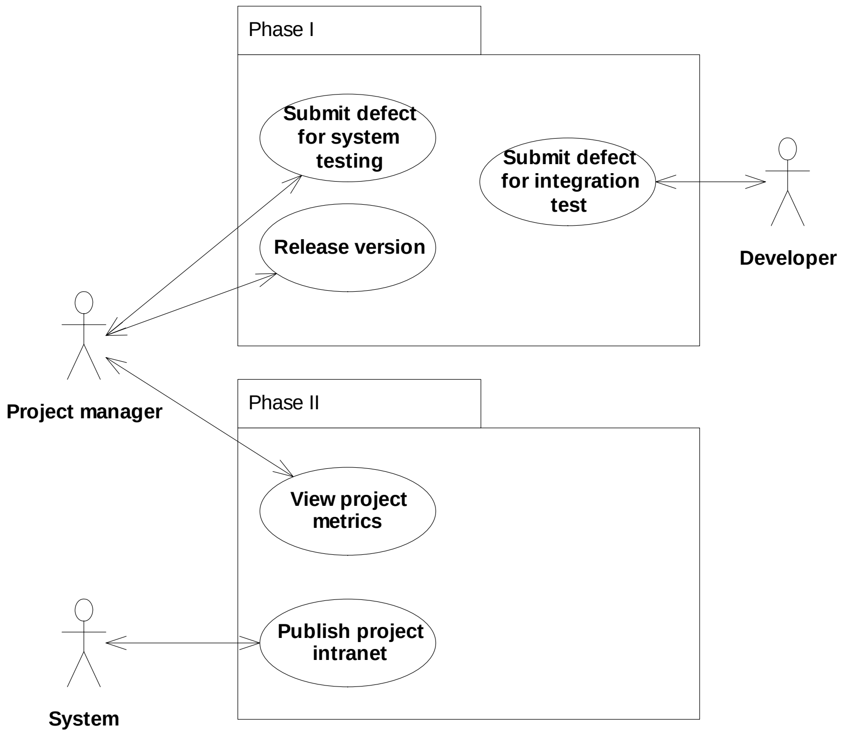
- It is complex in that it involves the interaction of several systems
- It is a part of the implementation that delivers significant business value
- It is important to the success of the implementation

Using these criteria we can identify the **key use cases** of the implementation. Investigating these use cases will feed into several aspects of the project planning process

- Scoping
- Estimation
- Security
- Documentation
- Training
- Identification of risks

Scope

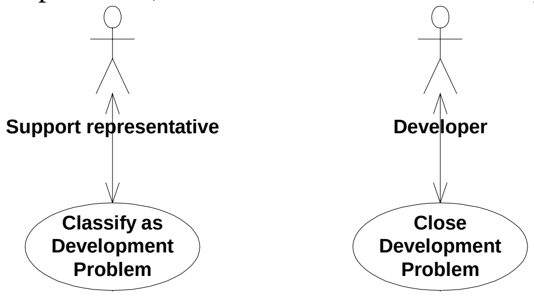
The diagram below shows a set of use cases grouped into a two phase implementation. As part of the scoping exercise we have identified the use cases that we will implement first and those that will wait until a later stage (the criteria used to make this judgement may be complex. However, the analysis of the use cases will give us the information we need to make an informed choice). Irrespective of any more detailed description of the individual use cases the diagram shows the scope of the project succinctly and clearly.



The development of all use cases to some degree of detail at an early stage is useful as it allows us to verify that the set is consistent and fulfils the requirements of the project as a whole, even if implementation is phased. It also demonstrates the understanding between the team, or vendor, implementing the system and the customer or stakeholders.

Estimation

The two use cases below summarise how we may integrate an enterprise help desk system with our development tracking system. A support representative raises a help request and further analysis reveals this to require a change to software by the development team. When the support representative classifies the request as a Development Problem we want to automatically raise a corresponding Defect in the development tracking system. This is then assigned to a developer and when completed we want to communicate this fact to the help desk. To improve visibility we could feed back more information about the development process to the Support Department, but for the sake of this example we'll just communicate completion.



Notice here we've set the scope; for this phase we'll only communicate completion. We could add more use cases such as Change Defect State to a later phase.

Elaborating each use case gives us help in estimating the time we'll need for these tasks.

Classify Help Request as a Development Problem

Actor: Support Representative

The support representative edits an existing Help Request. The Request Type is set to "Development Problem" and the request saved. The request must not already have a development problem associated with it.

A Defect is raised automatically in the development tracking system. The Request's ID is copied into the Defect and the Defect's ID is copied into the Request. The new defect is assigned to the Development department.

The request is held in a state of "On Development" until the Defect is closed.

Later in this paper we'll look at how we can decompose a use case in more technical detail. The use case rightly doesn't go into technical detail about how we're going to achieve the automatic creation of the Defect, but we do have enough information to identify interfaces with other systems and areas for automation.

Close Defect

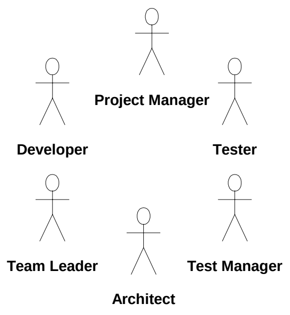
Actor: Developer

A developer has been assigned a Defect to work on. The developer has completed and tested the work and can close the issue. The developer sets the state of the Defect to Closed.

The state of the related help desk Request is automatically set to "Development Done".

Security model

Through developing the important use cases we identify the actors involved. The set of actors gives us insight into the groups of user we need to consider when we develop the security permissions for the system.



Each use case identifies the actor(s) that can execute the operations. However, as an adjunct to our model of the key use cases we can map our actors to more fine-grained permissions:

- The ability to edit data in a form or within the repository
- Visibility of data, say specific fields on a form or areas of a repository
- The ability to perform detailed operations not expressed by the key use cases (for example the creation of a baseline by, say, Team Leaders only).

Documentation and training

A very important aspect of the use case model we develop is that it is in non-technical language and uses the vocabulary of the business domain. As such they can be understood by everyone involved in the project.

- **Stakeholders**. Does the implementation fulfil the requirements of the stakeholders?
- **Training materials**. The key use cases usually express the more complex aspects of the system from an actor’s point of view. This feeds into the development of training materials. A technical author can elaborate the use case model to explain the system as individual interactions and as a whole.
- **Technical documentation**. A use case model is a central part of the development model within UML (see the 4+1 views of the Unified Process where the +1 is the use case model).

Identification of risks

During the elaboration of our use cases we will gain a better understanding of technical and organisational complexities. This can lead us to identify risks in several areas:

- People.
  This may involve the reluctance to implement processes or the ability two groups of actors to come to a consensus.
- Process.
  We could have difficulty in reconciling the requirements of the actors or integrating new processes with existing ones.

- System.
  This can be any technical risk, for instance the integration of systems or the automation of processes.
- Time.
  A particular use case may be required by a certain date to meet regulatory requirements or a business opportunity.
- External.
  A use case may depend on the delivery of functionality within a new version of a tool. Slipping of the third-party release will introduce delay.

The use case is the focus of discussion and the mechanism to record and track the risks. The use cases can also be fed into established processes for risk management used by the business.

Execution and Testing

Armed with our use case model we understand the functions of the system that are the most important as far as delivering business value and/or the most complex in terms of implementation. The customer and other stakeholders understand the system in terms of the key use cases and can see concrete progress as each use case is demonstrated.

The implementation of the system will usually consist of system configuration and customisation, and maybe the development of custom code to integrate with other business systems. Consequently, it may be the case that a use case contains enough information to allow it to be implemented. This is rarely the case in a traditional development scenario where most of the functionality to implement the use case will have to be coded from scratch. In the situation we are describing most, if not all, of the functionality will be available in the toolset we are implementing.

Where this is not the case we need an additional level of detail to provide confidence in the technical solution; confidence in our estimates and documentation for future maintenance.

Drilling into the use cases

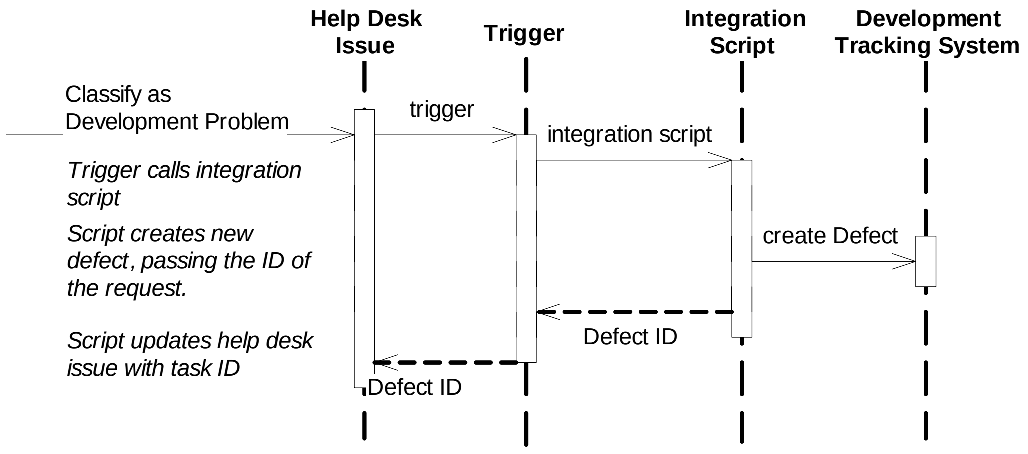
Again, we can look to UML to show how we can take a use case and develop its implementation. This is a large subject and it will only be explored briefly here. There are many resources available in the literature and on the web. It is common practice to decompose use cases into Sequence Diagrams. These diagrams show the interaction between system components in terms of the messages they send to each other. A *Collaboration Diagram* is a different way of expressing the same information. You may find one or the other more useful depending on personal preference. Sequence diagrams focus on the chronology of events and are useful when considering the synchronous or asynchronous processing of messages.

In sequence diagrams objects, subsystems or systems are represented by dashed vertical lines. The messages between systems are shown as horizontal arrows that may be annotated with message details. Time runs from the top to bottom. The left of the diagram can contain explanatory text.

The two help desk use cases above will illustrate the use of sequence diagrams.

Classify as Development Problem

Here, the Support Representative performs the operation to classify the help desk request as a development problem, probably by setting the value of a field. The help desk system (via a trigger) must then invoke an integration script that creates a corresponding issue in the development tracking system. The unique identifier of the development defect is passed back by the script to the help desk system.

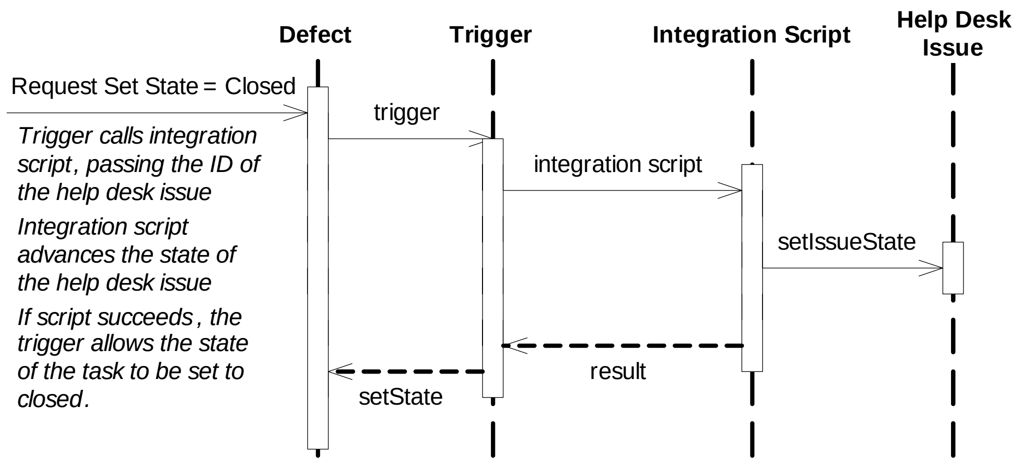

Help Desk Issue
Trigger
Integration Script
Development Tracking System
Classify as Development Problem
trigger
integration script
create Defect
Defect ID
Defect ID
Trigger calls integration script
Script creates new defect, passing the ID of the request.
Script updates help desk issue with task ID

This quickly identifies the tasks and deliverables to implement the use case.

- Create a trigger in the help desk system that fires when the classification field is set.
- Create an integration script to create the development task.
- The integration script must have a mechanism to update the help desk issue with the ID of the development task.

Close Defect

When the developer closes the task we have a similar sequence of operations. Note that in the previous integration we have recorded the identifier of the help desk issue in the development task so the second integration script can invoke the set state operation on the correct issue.


Defect
Trigger
Integration Script
Help Desk Issue
Request Set State = Closed
trigger
integration script
setIssueState
result
setState
Trigger calls integration script, passing the ID of the help desk issue
Integration script advances the state of the help desk issue
If script succeeds, the trigger allows the state of the task to be set to closed.

For simplicity the diagram does not explore the failure path.

Conclusions

Although traditionally used within software development projects we can use the technique of use cases when we come to deploy Application Lifecycle Management systems. ALM systems usually touch many parts of the business within and beyond the development sphere so it is important we understand the scope of the implementation and prioritise its activities in terms of business value.

The non-technical, problem domain vocabulary used means all the stakeholders can understand and contribute to their development aiding communication and removing ambiguity. The model also feeds into other aspects of the implementation such as the detailed design, planning, training and documentation.

References

Use Case Driven Object Modelling with UML.
Doug Rosenberg & Kendall Scott.
Addison Wesley, ISBN 0-201-43289-7

Developing Software with UML.
Bernd Oestereich.
Addison Wesley, ISBN 0-201-75603-X

Software Configuration Management Patterns.
Stephen P. Berczuk, Brad Appleton
Addison Wesley, ISBM 0-201-74117-2

Adopting an Agile Method

Alan S. Koch
http://www.ASKProcess.com

Abstract

The argument has been made: “We should be using an Agile software development method.” And the command has rung out: “Make it so!” Now what do you do? How do you take that one-line “requirement”, and make it so?

Adopting an Agile method is no different from any other change we might make to the methods and tools we use. We must determine why we are embarking on this course, choose the method that will satisfy the need most closely, then map out the path from where we are today to where we need to be. *Then* we can “make it so”.

Why Adopt an Agile Method?

Before we start changing things, lets first step back and be sure we understand what it is that we are trying to fix. After all, if there were not a problem with the way we do things today, there would be no reason to change. So, our starting point is the original argument for adopting an Agile method.

If there was significant discussion before the decision to adopt was made, then the information we are looking for may already exist. If little was discussed (or if there is no record of what was discussed), then you will need to work with the person who suggested the change to understand why it was suggested. Are there problems with the methods we use today? What are they? How bad are they? Who is affected by them? Or is there the expectation that although no large problem exists, an Agile method would allow us to improve over our current methods? What sort of improvement is anticipated? How significant is it expected to be? And who might be the beneficiaries of the improvement?

We also need to discuss with the decision-maker why they decided to approve the idea. What did they envision? What benefits do they expect? How do they think things will change as a result of adopting this new method? And most importantly, what would be this person’s definition of “success” in this endeavor?

Of course, when we discuss these things with multiple people, we may find that there has not yet been a meeting of the minds on exactly why we should adopt an Agile method. In that case, we must bring the diverging minds together (preferably in the same room) to discuss their different perceptions and agree on exactly why we want to do this.

Regardless of what it takes, we *must* end up with a clear and consistent description of the reason why we are adopting a new way of working. Indeed, without this, we are virtually assured of failure, and some of the major constituencies are likely to be unhappy with the end result. Conversely, with a clear understanding of the “why” of the effort, we are equipped to move forward with confidence.

Adopt Which Agile Practices?

Now that we know what we want to achieve, we can examine the various practices that we could adopt and decide which of them will be appropriate in our organization. This is a piece of the process that is easy to miss, especially if the mandate is to adopt a specific Agile method (like eXtreme Programming or Scrum). But even if a specific method has been prescribed, it may not make sense to adopt *all* of the practices of that method. We need to carefully consider the uniqueness of our organization and the projects we undertake to decide which practices will fit in, and will help us in achieving the goals we have identified.

The major areas that we must consider are:

- Our organization's culture
- Our customers and how they prefer to interact with us
- The types of projects we do
- The tools and processes that we currently use
- The strengths and weaknesses of our software-related staff

Organizational Culture

All of the Agile methods are built around the concept of a self-directed team. The development team is not told what it will do and when it will be done by. Rather, they are given broad direction about goals, and then they work with the customer to determine how to reach those goals.

If your organizational culture is build on a “Command and Control” style of management, then many of the practices of the Agile methods will represent a serious challenge to the organization. The managers will need to alter their management methods and adopt a more collaborative relationship with the development teams. This is a very difficult transition for many old-school managers to make, and could be a significant stumbling block to the successful adoption of *any* Agile practices.

Another organizational issue revolves around planning and commitment making. If your organization expects plans and requirements to be finalized up front, and then adhered to throughout the project, then the Agile methods’ incremental planning and requirements definition practices will be problematic. The Agile methods assume that we cannot know all of the things that will come up during the project, so they plan and define requirements at only a high level in the beginning. Then they add detail and revise them incrementally throughout the project, always keeping the project’s broad goals in sight.

Examine the practices that you expect to adopt, and for each, determine the extent to which it can be implemented given the realities of your organization’s culture, and the benefits you anticipate can be *reasonably* expected to accrue.

Customers

All of the Agile methods expect significant participation of the ultimate customer of the project with the development team throughout the project. This is often accomplished by having a representative of the customer working closely with the team on a regular basis. (Different Agile methods carry this model to various extremes – with eXtreme Programming being the most extreme.)

How active are your customers in your projects now? And more importantly, how readily would they increase their participation if you asked them to? Most customers have only limited resources to devote to the project, and any significant increase in interaction could represent an obstacle to them. Would your customer be able to commit the time and effort that the Agile practices expect of them?

Another consideration is the details of your contract with your customer. If you are developing software under contract, then the rules for interaction may be carved in stone and essentially unchangeable

Even if changing the rules of engagement were possible, would your customer be willing to do so?

Often, companies use contracts to protect themselves from the organization they are contracting with. If your customer takes this view, then they may not be *interested* in collaborating more closely with your development team.

Examine the practices that you expect to adopt, and for each, determine the extent to which it can be implemented given the realities of your customers, and the benefits you anticipate can be *reasonably* expected to accrue.

Projects

Every project is unique and represents unique challenges and opportunities. Nonetheless, most organizations tend to take on projects that have a relatively predictable set of attributes. Some companies do mainly financial and administrative systems. Others do real-time embedded systems. What kinds of projects to your development teams generally work on? The types of projects that your teams take on have a certain amount of uncertainty to them. They represent a certain risk profile. And they include a certain amount to technological innovation.

Although the Agile methods were mainly developed to meet the needs of small projects that expect significant turbulence and change, they can also be used in many other kinds of environments. Most of the Agile methods expect that the development team can work face-to-face on a daily basis. If this is not the case in you organization, then some of the Agile practices may need to be modified to work with a distributed team.

The promoters of the Agile methods tell us that their methods can and should be used on *any* project. Indeed, some of them are experimenting with Agile distributed teams. While it may be true that Agile methods *can* be used on any project, the questions you must ask is, *should* you use the practices these methods prescribe on *your* projects!

This means going back to the purpose you identified for adopting an Agile method, and determining if the practices will support achieving those goals. Examine the practices that you expect to adopt, and for each, determine the extent to which it can be implemented given the realities of your projects, and the benefits you anticipate can be *reasonably* expected to accrue.

Tools and Processes

The methods we employ do not exist in a vacuum. They are strongly influenced by the environment in which we use them; and an important part of that environment includes the supporting tools and processes that we depend upon.

As things stand today, your organization already has a set of tools and processes in place. To what extent will those tools and processes be compatible with and support the new Agile practices you expect to adopt? Will your processes and tools (e.g. for requirements management, or for Quality Assurance) stand in the way of adopting Agile practices? And if so, is there latitude to change them (or get rid of them) in order to make the environment more conducive to the Agile practices?

The flip side is also true. Many of the Agile practices require specific tool and process support in order to be effective. For example, eXtreme Programming expects that automated testing tools are available to the entire programming team, and that they use them heavily. Are such tools available in your organization? And if so, are there enough licenses to allow this widespread use?

Another example is code control tools. Most of the Agile methods assume strong code control systems and strict adherence to the disciplines involved in using them. Liberal use of things like refactoring and shared ownership of code can only work in an environment that provides the necessary code control tools and processes.

Examine the practices that you expect to adopt, and for each, determine the extent to which it can be implemented given the realities of your tools and processes, and the benefits you anticipate can be *reasonably* expected to accrue.

Staff

As we noted earlier, the Agile methods expect that development teams will be self-directing. This means that instead of being told what to do, the team understands the objectives of the project, and they collaborate with the customer and each other to determine the most appropriate steps to take at each juncture.

In order to act autonomously, the team members must be willing and able to take on a new level of responsibilities and accountability. Although this may seem to be an easy step (based on the complaining we often hear from programmers), it is actually quite a leap to go from complaining about the status quo to taking on the responsibility for creating a new one!

Many of our programmers are not as bold as we might expect. They may have good ideas about things to change, but be unwilling to take on the accountability that comes with being a decision maker. To many folks who thrive on technical challenges, the messiness of project and customer challenges can be quite intimidating.

And of course, our development teams do not represent uniformity in capabilities. Some of our folks are indeed super stars! But others' abilities are quite lacking. If we are honest with ourselves, we must admit that the average programmer on our teams is only average! And around half of them are below average.

The Agile methods empower teams to direct their own work (in collaboration with the customer). It is true that as they work in this environment, our people will learn and grow in their ability to deal well with making project decisions. But how successful can they be at the beginning? And at what point will they reach their maximum potential?

Examine the practices that you expect to adopt, and for each, determine the extent to which it can be implemented given the realities of your staff today, and the benefits you anticipate can be *reasonably* expected to accrue.

Adopting an Agile Method

After we have considered all of the things we just discussed, we will be equipped to decide which Agile practices would be appropriate in our organization, given our culture, customers, projects, tools, processes and staff. Armed with this information, we can evaluate each Agile practice against our objectives, and determine which ones we should adopt, and if we need to customize them in any way.

In essence, we will be “rolling our own” Agile method. But isn’t that what we need to do? We don’t want to adopt something that might have worked in some other organization. We want to adopt what will work in *our* company. We want to do the right thing so we can achieve our goals in this adoption, and improve our ability to do a good job of building good software.

The Cornerstone of a Great Shop

Jared Richardson
http://www.jaredrichardson.net

I did a lot of lawn mowing when I was younger. My brother and I tried to make our summer money by asking real estate agents if we could mow the lawns of their absentee clients. We'd usually land one realtor each year and they'd give us enough business to keep us busy all summer. One of the things I learned is how hard it is to cut a straight line when you're mowing a wide yard. When I was in the middle of the yard, I felt like I was cutting a straight line, but then I'd get to the end of the row and look back to discover a crooked line. It always amazed me how something could seem so right and be so off course.

Timely Feedback

A software project can be a lot like mowing a yard. Even though we try to move in a straight line, and we think we are, later we look back and are amazed at how far the project ran off course.

Whether mowing yards or building software, we need timely feedback to help keep us on track. Looking back at completed software projects, or lawns, shows you where you missed the mark, but it's usually too late for that project. We need feedback while we're still in the midst of the work. I never found a good way to get that feedback for my lawn mower, but I have found a guide for software projects. I use continuous integration systems to keep my projects on track.

Continuous Integration

Mike Clark calls this type of system a "virtual build monitor". This extra team member keeps an eye on your project and lets you know when things start getting off course. If you invest in a good automated test suite, you'll quickly catch all sorts of errors that traditionally pull good projects off course.

The more shops I get to observe, the more I'm seeing that continuous integration (CI) plays a vital role in keeping a shop on course. In fact, these days I'm telling people that I've learned one of the basic, fundamental principals to keep both you and your project on target.

Do you want to make your product a great one? Do you want to be the best developer you can be? Then make a solid continuous integration system a first-class member of your team and the cornerstone of your shop. A good CI system eliminates many of the problems that prevent you from working on the product, your career and your craft.

A continuous integration system does several things automatically.

- Monitors your source code
- Compiles after every change
- Tests your compiled code
- Notifies the developers of problems as soon as they occur

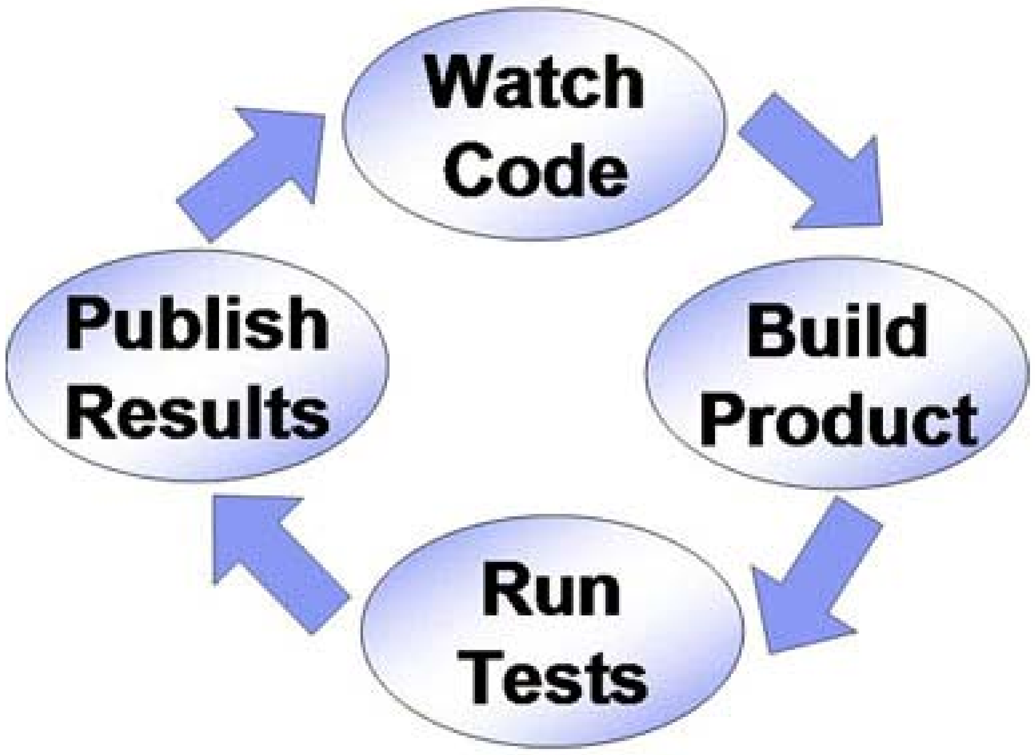


Figure 1. Continuous integration actions

As we move forward, keep an open mind and try to see where each step could’ve helped you in the last few months. Then, when we’re done, I’m going to point you to a Continuous Integration system that is trivial to install, easy to use, and open source to boot.

Let's look at what a continuous integration system is and why it helps so much.

The Steps of Continuous Integration

Continuous integration systems all have a few common steps.

First, CI systems monitor your source code. The system usually watches your source code management system (CVS, Subversion, Perforce, ClearCase, Visual Source Safe, etc) but most systems can also monitor other resources, like file systems. This is how the software knows it's time for a build. Every time your code changes, the CI system checks out the latest version of your code.

Second, the software compiles your project. The system runs your existing build scripts by wrapping them in an Ant script. In this step, your CI software is requiring you to have a scripted build. If you builds are not robust or repeatable, your CI tool will expose this flaw. It will force you to have a clean build system.

Third, CI systems tests yours new build. The tests are created (or wrapped) in an Xunit framework (Junit, Nunit, HtmlUnit, jsUnit, etc), which means you have access to dozens of test frameworks that range from unit testing to browser click through testing. When you set up a system to run tests, people are more likely to write the tests. They'll also contribute the tests they've been hiding on their own machines.

Lastly, your CI system will notify everyone of the results. The developers or testers who just changed the code will get email telling them how long the build and test took, how many tests passed, how many failed, etc. Your system will also archive the results to a web page.

However, the publishing step is very configurable. You can publish in a variety of interesting ways beyond a standard web page. You can publish to a custom web page, XML log, email, an instant messaging client, or even a Lava Lamp. The publish step is an extremely flexible way of sharing your build results.

What's the big deal?

There are several key practices that continuous integration either requires or encourages. They are source code management, scripted builds and test automation. Much of the benefit that comes from using a CI system actually comes from the foundational practices that a CI system requires.

Don’t get me wrong. CI adds plenty of benefit as well. It’s just that many day-to-day problems go away when you use these other practices first.

Code Management

One of the first things your CI system will do for you is make sure you've got your source code organized and (hopefully) into a source code management system. After all, your CI software can't watch a code tree you can't identify. The first practice CI encourages is good source code management.

This benefit will seem very elementary to many people, but I've seen shops that still use network drives and zip files. Quite a few developers still haven't discovered source code management.

Proper source code management doesn't take much time at all once you've learned how to use it. Like any good tool, you’ll save much more time than you’ll spend learning to be effective with the tool.

You’ll save the time you normally spend reconciling code differences by hand, not to mention rewriting the work that careless coworkers overwrite from time to time. Code collisions and lost work are common issues but a good source code system also merges your changes for you, maintains a history for each file, and more.

If you're not using a proper source code management system, I urge you to rethink you position. It's a huge time saver.

A Scripted Build

The second thing your CI system will require is a scripted build. Moving to this step requires a level of build automation. Fortunately, this is easy to add. There are many tools available, both commercial and open source, that solve this problem for you. You still have to understand how to build your product, but these tools will keep from learning the different command line options for javac or jar on different operating systems. Look at tools like Ant, Maven, and Rake.

Like good source code management, a scripted build provides many benefits.

For starters, your teammates aren't all busy building their own version of the build script. Everyone needs to build and developers, being clever, will all find a slightly different way to solve the same problem. When you have a single build script, everyone's building the same way. It's okay is someone still wants to build differently (an IDE maybe?), but they need to have the ability to build the same way that everyone else does.

Don’t ignore the maintenance savings either. You'll eventually improve the build script, find a bug in it, or decide to make it faster. With a single script, you do the work once time. When everyone has his or her own build method, everyone solves the same problem repeatedly. What a waste of time!

When you build your code the same way, everyone gets the same version of the product. This means that the testers report problems in the same version of the program the developers run.

Without a canonical build script, you don't always get everyone on the same page. In fact, the customers, testers and developers often run very different versions of the same product and then wonder why they can’t reproduce the same issues. If you’ve had trouble reproducing your customer’s bugs, then start here. Is everyone running the same version of the product?

Test Automation

Another practice that a CI system encourages is test automation. Writing and running tests is a huge milestone for many shops and is one of the hallmarks of a great shop. I think test automation is the core of why a CI system adds such benefit. People who recognize the benefit of automating common tasks tend to be more strategic thinkers. They automate everything possible, including building and testing, and it frees them up for more interesting work. (Of course, this doesn’t eliminate manual testing, but that’s another topic.)

What is an automated test?

- Binary
- Automatic
- Repeatable

Binary

A test with a binary result passes or fails unambiguously. There’s no question about whether the test succeeded. Sometimes a test will return a result that requires a judgment call from a tester. The odds are good that you don’t need this.

Work hard to make your tests clean and binary. Write them so they evaluate the result and tell you if it passed or failed.

Automatic

If the test isn't automatic then someone has to set up an environment, start the test, click a button, or look at the results. When this happens, the test becomes interactive again. Much of the benefit of test automation is lost.

You've created a hybrid test somewhere between an interactive test and an automatic test. Instead of letting a small number of testers baby-sit a large number of tests and continually adding more tests, you'll have a large number of testers looking at log files all day long. Half-automated tests are certainly better than pure interactive testing but they fall far short of where you can be. Work hard to make your tests completely automatic, including the determination of the pass or fail status.

Repeatable

An automated test also needs to be repeatable. A good test doesn't give you different results for three out of five test runs. If your tests aren't repeatable, break the tests down into smaller tests. Eventually you'll isolate the problem area and as a bonus, you’ll have new tests created for your test suite.

Don't forget about external dependencies either. You can rebuild and restock database tables cleanly before each test run with tools like Ant's SQL task or dbUnit (http://dbunit.sourceforge.net). A dirty database table can introduce all sorts of variation into a test run. (You may want to create a small but representative data set to load for your testing runs.)

Leverage Yourself

An automated test is a great way to leverage your experience and expertise. As an expert on your product, you probably know how to test parts of it in a way that few other people can. You can encode that knowledge in a reusable format by creating an automated test. This makes your experience available without diverting your attention. Sometimes a co-worker will run the tests, other times you will. In other cases, a program will run them.

Let these bits of your expertise exercise the product while you do other things, like go home on time, or stay late and solve problems that are more interesting. These tests might run while you are coding or at home sleeping, but you are doing something else. Your tests are working in the background.

Getting Started

Sometimes people won’t install a CI system because they don't have tests ready to run in the system. There are enough benefits from fast compile cycles to justify using Continuous Integration, so don’t wait. You don’t wait to see a doctor until you’re not sick anymore, right? Having the CI system keep your compiles clean will free up some of the time needed to start writing tests as well.

I've also found that people are much more likely to write automated tests if they’re sure the tests will be used. By providing a CI system, you have a place to house your tests and run them immediately. This is the best way I know to encourage test creation. People want to create things are used and this assures them the tests they create will run regularly.

The best way to get started with Continuous Integration is to start using an existing software package. I'm going to point you to CruiseControl on Source Forge (http://cruisecontrol.sf.net/). Since version 2.3 Cruise Control comes with an embedded servlet engine (Jetty-http://jetty.mortbay.com) and a sample project. You can download the project and see CC running in less than five minutes. Then, to add your own project, just copy the bundled example. It’s very easy to get started.

The CC team has a great write-up on how to run the binary release of CruiseControl. Visit http://cruisecontrol.sf.net/ and click the “Getting Started” link on the left.

In Conclusion

The teams I know running smoothly and cleanly always have continuous integration in place. It's a practice I respect more everyday.

I’m seeing this practice overshadow all others. Teams that run smoothly use continuous integration. They respect the system instead of tolerating it, and the developers treat the notifications seriously. When the system says something is broken, these teams address the problem quickly. These teams insist on CI coverage from the first day of a new product.

Other shops, even those who are using a CI system but ignoring it, are very different. They live in turmoil. Heroic efforts are not the exception but the rule. In fact, these teams always seem to be running behind. They always have a crisis issue to resolve or deadline to meet.

They work and live in a perpetual twilight of stress and problems. They’ve lived there so long that they think it’s the only way to write software. Sadly, these teams tend to burn people out. I’ve been there and it’s no fun. Creating software can be a great joy -and is- when done right. CI can’t solve every problem, but it can remove several categories of problems that would otherwise clutter your day and slow you down.

If you’re not using a Continuous Integration system, try one out this week. Get a system installed and leave it running for one month. At the end of that month, turn it off if you don’t see the benefit.

Don’t be surprised if you find yourself missing the system the first day it’s gone. You might just become one of the developers who insists on continuous integration coverage on your new projects.

Resources:

- Martin Fowler and Matthew Foemmel on Continuous Integration http://martinfowler.com/articles/continuousIntegration.html
- CruiseControl page http://cruisecontrol.sourceforge.net/
- CI product page: http://www.jaredrichardson.net/ci.html
- Scripted build links: http://www.jaredrichardson.net/buildscripts.html
- Mike Clark’s automation blog: http://www.pragmaticautomation.com
- Jetty http://jetty.mortbay.com/

AdminiTrack offers an **effective web-based issue and defect tracking application** designed specifically for professional software development teams. See how well AdminiTrack meets your team's issue and defect tracking needs by signing up for a risk free 30-day trial.

http://www.adminitrack.com

ObjectiveView issue 9 is now available at www.ratio.co.uk/objectiveview.html Issue 9 contains articles on AspectJ, AJAX, Ruby and Ruby on Rails, + opinion pieces by Rebecca Wirfs-Brock, Scott Ambler and Ken Pugh.

Interested in training in Ajax, UML or Ruby/Rails - visit www.ratio.co.uk for full details.

Advertising for a new Web development tool? Looking to recruit software developers? Promoting a conference or a book? Organizing software development training? This space is waiting for you at the price of US $ 30 each line. Reach more than 39'000 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 25'000 visitors/month of our web sites! To advertise in this section or to place a page ad simply go to

http://www.methodsandtools.com/advertise.html

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