
METHODS & TOOLS

Practical knowledge for the software developer, tester and project manager

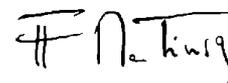
ISSN 1661-402X

Summer 2012 (Volume 20 - number 2)

www.methodsandtools.com

Do Software Developers Want to Be Managers?

According to surveys, Agile is now implemented by a majority of organizations and Scrum is the project management framework the most widely used. A characteristic of Scrum teams is that they are (or should be) self-organized. Team members are now responsible to divide between themselves the workload and they will need to deal with conflicts between team members. The former project manager authority and responsibilities are spread amongst the Scrum team. But do Agile developers really want this? Self-organization relies on the ability of team members to communicate and negotiate. The transition from a hierarchical team organization to a flat relationship where everybody participates is not easy and faces many challenges. Software developers have in majority an introverted personality type and this is not the best setting to interact with colleagues or even just to express needs or feelings. Teams might end up as just being organized by the most vocal members. Another implied hypothesis of self-organization is that all team members should have an equal weight in decisions, even if technical superiority or leadership might be recognized. This might not exist when you have teams that mix internal employees and contractors that might be denied authority. Self-organization also works in a culture of individual responsibility, affirmation and (positive) contradiction. This is not always the case in all organizations or countries. The team has now to deal with non-performing members, people that are either technically limited or that have a difficult personality. Team members have now to negotiate with these people to try to use them as best as they can or initiate the process that will remove them from the team. As we might have wished sometime to have the power to fire somebody we were working with, having to do it for real is something completely different. The self-organization of Scrum gives more power to team members, but with the power also comes the responsibility. The Agile transition from traditional project management is not easy and this type of team organization might not be the best for all people or culture, local or organizational. I am not sure that all Agile developers want to bear some management responsibilities. But being Agile doesn't mean that development managers have to stop working to maintain healthy relationships between team members. As the ScrumMaster, but from outside the team, they still have to be ready to provide support and actions to keep the project on track, walking on the thin line between letting the team grow by itself and preventing issues to deteriorate too seriously.



Inside

Creating an ATDD Ready Sprint Backlog	page 3
Continuous Delivery Using Maven.....	page 9
DSDM Atern Overview	page 23
Knowledge Management and Software Organizations	page 33
Erlang Open Telecommunications Platform	page 45
Concordion - Automated Acceptance Tests in Java.....	page 49
Mockito - Mocking Framework for Java	page 57
Robotium - Android Test Framework.....	page 62
JMeter-Plugins - More Obvious and Powerful Load Testing	page 65

SpiraTeam the complete Agile ALM suite - Click on ad to reach advertiser web site

Are You Tired of Having Separate Tools for Requirements, Testing, Bug-Tracking and Planning?



It's time to try a better way.

spiraTeam[®]



The most complete yet affordable
Agile ALM suite on the market today.

Learn more at: inflectra.com/spiraTeam

www.inflectra.com
sales@inflectra.com
+1 202-558-6885

inflectra[®]

Creating an ATDD Ready Sprint Backlog

Ralph Jocham, effective agile, <http://agiletips.blogspot.com>

Scrum is a very powerful framework to drive out the right requirements and thereby delivering the best possible product in a given time frame. This is made possible by frequent inspection and adaptation based on ongoing transparency. One of the empirical process-control-feedback loops is the Sprint Review in which the product increment is being made public for review and feedback. This feedback helps to discover new features, remove unneeded ones and alter them based on the gained insight. It is nothing less, but effective just-in-time planning to maximize the ROI. Most of those new features are broken down into requirements during the Product Backlog grooming sessions and subsequent Sprint Planning meetings.

Scrum and the Scrum Guide do not mandate how you write your requirements, many prefer User Stories but any other format is ok as well. The same applies for the Sprint Backlog, Scrum only requires that you have a plan to deliver the selected Product Backlog items and that at any given moment during the Sprint you know how much work is remaining visualized with the Sprint Burndown, another popular Scrum practice. As you can see, Scrum is a rather minimal framework, with only a few strict rules and many possibilities to apply proven practices in your given context.

Sprint Backlog

As mentioned before, the Scrum Guide defines the Sprint Backlog as the set of Product Backlog items selected for the Sprint plus a plan for delivering the Product Increment and thereby realizing the Sprint Goal. One possible enhanced practice of a plan can be to specify the Sprint Backlog with a set of examples. It is a rather new practice, but gaining more traction in recent years.

With this approach you create an extended implementation of a Sprint Backlog that responds to two fundamental questions:

- How can the Development Team be sure that the plan is adequate in fulfilling the underlying requirements?
- How can they foster collaboration and confirm repeatedly on a recurring basis that the software under construction always meets the evolving requirements?

Planning and driving a Sprint with examples allows the Development Team to better understand the requirements. It seeks to add an extra level of conversation to eliminate the communication gap between business and the team. This is done by illustrating each Product Backlog item with a cohesive set of concrete examples, expressing the success criteria. These examples, which are nothing less than a provable plan, will guide the Development Team to build the right product right. During the Sprint, by translating these business-facing examples into automated tests, we gain more transparency about the business need and simplify the “inspect and adapt” cycle.

For many of us, it is assumed that the plan of the Sprint Backlog is a list of technology-facing tasks ignoring the ‘why’ in the first instance. The underlying assumption is, that when all tasks are completed, the Product Backlog item will be ‘done’ right.

tinyPM the smart Agile collaboration tool - Click on ad to reach advertiser web site



**NEW
HOSTED
VERSION**

**FREE 5-USER
Community Edition**

DOWNLOAD

<http://www.tinypm.com/download>

Tiny Effort, Perfect Management

Web-based, lightweight and smart agile collaboration tool with product management, backlog, taskboard, user stories and wiki.



Team collaboration

tinyPM let you focus on your project and the team by making all boring mechanics invisible.



Customer engagement

Great adoption within business leads to better project awareness within the whole team. Translated into 16 languages.



Agile management

tinyPM makes sure that all team members, clients, stakeholders feel comfortable with your agile process.

Integrations

tinyPM gets data from bug trackers, mail and more, so you can have all the information you need in one place.

Integrates with: Git, Mercurial SVN, Bitbucket, Github, JIRA, UserVoice, POP3, RSS/Atom

Features

General

- Local (on-site) installation and hosted edition
- Multiple projects support
- Advanced permission management
- Timezones
- Localized (16 languages)

Main functions

- Dashboard with cross-project view
- Sandbox (feature requests/ideas/bugs)
- Backlog (Drag'n'Drop)
- Taskboard (Drag'n'Drop)
- Timesheet (time and budget tracking)
- Wiki
- Activity history

Release Planning

- Grouping iterations into releases
- Release delivery forecasts

Iterations

- Iteration planning and tracking
- Iteration goals

User stories

- Estimation with customizable scale
- MoSCoW priorities
- Splitting user stories
- Automatic work progress
- Tags
- Acceptance
- Card colors
- Changes history (versioning)
- Attachments
- Comments
- Printing cards

Tasks

- Progress
- Converting tasks into stories
- Moving tasks between stories
- Multiple users assigned to a task
- Estimation with customizable scale
- Changes history (versioning)
- Attachments
- Comments

Metrics

- Project burndown/burnup charts
- Budget burndown charts
- Iteration burndown charts (stories and tasks)
- Velocity chart
- Backlog progress display

Extensions

- E-mail notifications
- RSS feeds
- REST API over HTTP
- Plugins
- Stories imports & export (CSV)

Now with **Customizable Taskboard!**

www.tinypm.com

tinyPM is advanced, lightweight and smart tool for agile collaboration including product management, backlog, taskboard, user stories, wiki, integrations and REST API.

tinyPM goes beyond software development and encourages all team members (including clients, management and stakeholders) to actively participate in all your projects.

Contact us

support@tinypm.com

Follow us

twitter.com/tinypm



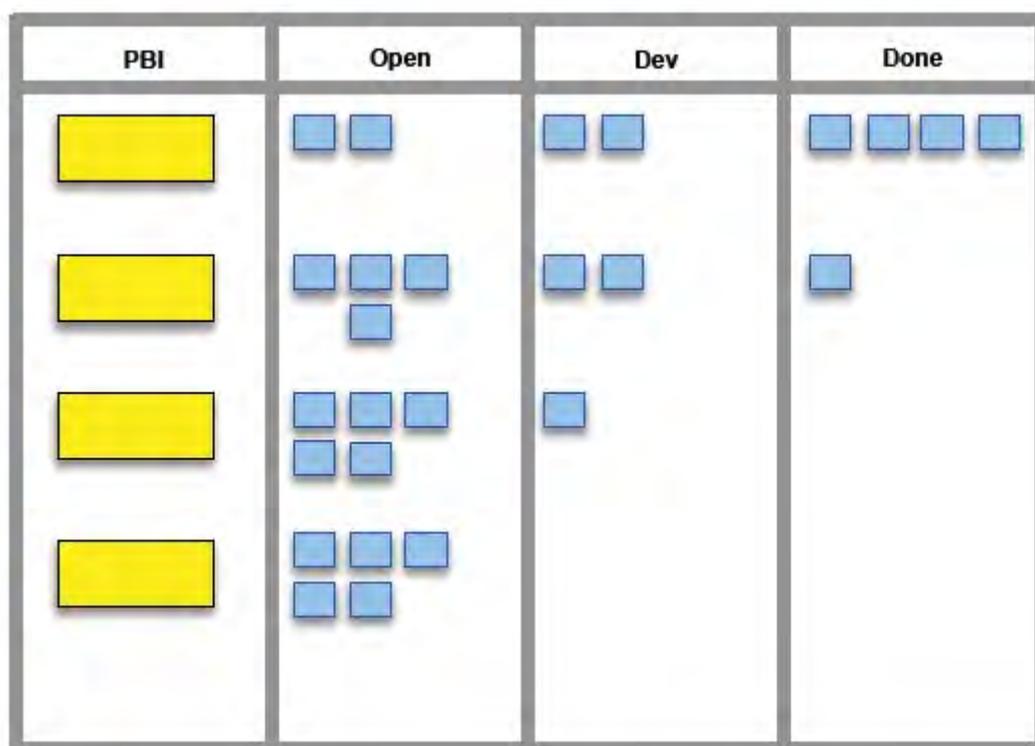


Figure 1 - Classic Sprint Backlog

Splitting requirements into tasks can help developers to think about the amount of work involved and leads to more accurate estimates but it is a poor strategy to foster collaboration. In the end, the progress during the Sprint is measured with the completed Product Backlog items. A Sprint Backlog that is centered on the validation of business needs helps to be more effective in delivering the business value right.

The underlying Problem

Doing the right thing and doing the right thing right - this is the paramount problem of software construction. Even though Scrum helps trawling the right requirements through ongoing Product Backlog grooming, almost invariably there is a communication gap, a misunderstanding between Product Owner and the Development Team. The reasons for this is that requirements are ambiguous, tend to be abstract and often omit the ‘obvious’ details. Therefore, they are hard to grasp, not tangible and cannot easily be projected into a cohesive solution. Furthermore, if we purely plan the Sprint with a list of tasks that ignore the channels of communication with stakeholders, this adds risks that can cause too much Sprint Backlog variation and make it hard for the team to reach the Sprint Goal.

The described practice suggests an alternative strategy to plan the Sprint. This has major impacts affecting the Sprint Planning Meeting, the Daily Scrum, Sprint Review and the Grooming if you choose to do it. During the Sprint Planning/Grooming, the plan produced by the Development Team is augmented with a set of concrete examples illustrating each business requirement. Because the work is now also expressed in terms of business requirements, during the Daily Scrum, each team member can easily focus on the bigger picture and align with the ‘why’. There is no lack of awareness of the Sprint Goal. Furthermore, by translating the business-facing examples into automated tests, it enables the Development Team to systematically and quickly inspect during the Sprint that the software increment always meets the evolving requirements towards the Definition of Done and the overall goal.

Automation

Assisted by the business-facing automated tests, the Development Team can easily confirm that all examples work successfully. This changes the dynamic of the Sprint Review because the conversation is much more about the future requirements than the work done during the Sprint.

We apply the practice in two stages:

- During the Sprint Planning Meeting and/or Grooming, the Development Team illustrates each Product Backlog item with a cohesive set of examples and related data in order to create a plan. These examples are concrete, tangible, provable and automatable.
- During the Sprint, the Development Team translates those examples into tests in order to inspect the Increment under construction. These tests should be automated to facilitate fast regression testing.

In addition to the regular tasks, each Product Backlog item selected for the Sprint is augmented with a set of examples and each one illustrates the expected behaviors. These concrete examples communicate and validate the acceptance criteria.

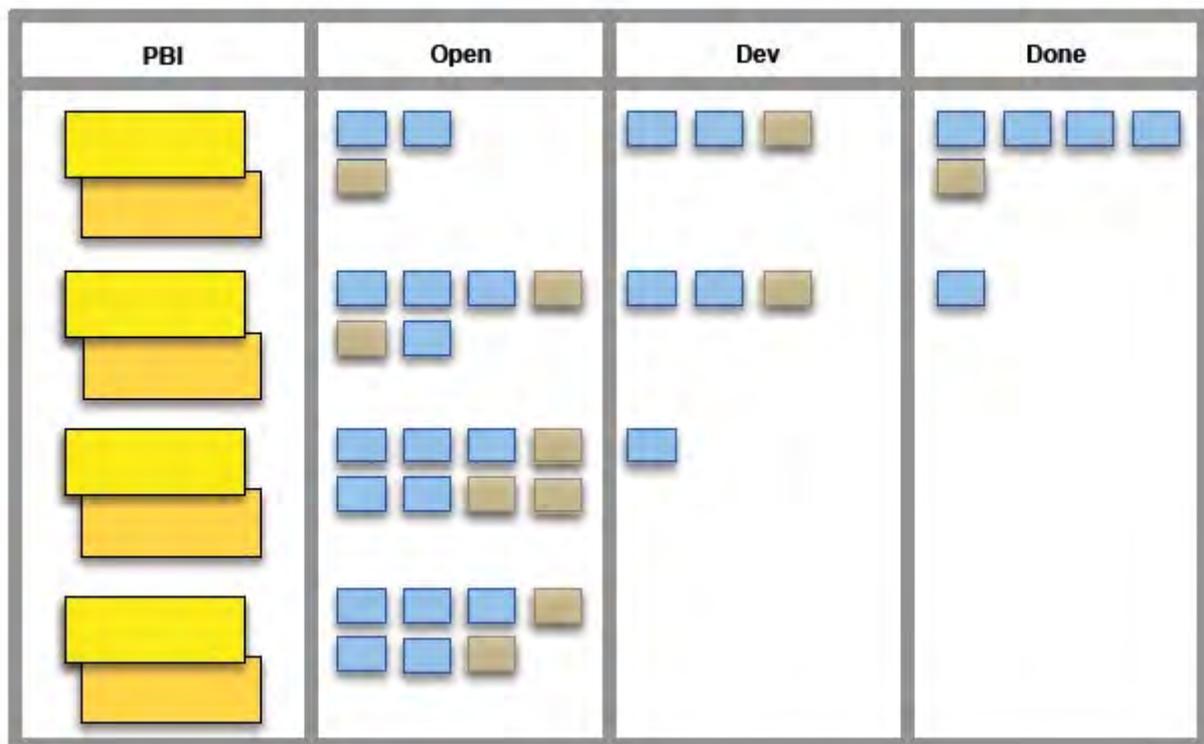


Figure 2 - Sprint Backlog specified by Examples, ready for ATDD. The dark yellow PBIs (flip side) contain additional examples and related test data. The orange-brown tasks are ATDD related work items

The Sprint Backlog, enhanced with a set of examples in form of automated tests, is a highly visible, real-time picture of the work that the Development Team plans to accomplish during the Sprint. These concrete examples provide enough detail so that changes in progress can be understood throughout the Sprint, specifically during the Daily Scrum. If a new example is required, the Development Team adds it to the Sprint Backlog. When an example is added, the estimated remaining work for the whole Sprint is updated.

By translating the business-facing examples into tests, we enforce an additional level of communication, which answers the question: ‘How can we prove, that the Increment is right and working correctly?’ It is this conversation that puts the whole Scrum Team on the same page.

Planning and inspecting with examples will fundamentally improve the conversation between the Development Team and the Product Owner. Card, Conversation and Confirmation as proposed by Ron Jeffries is put to it’s best. As a result of correctly implementing this practice, no Product Backlog item should be rejected by the Product Owner during review without good reason as not "Done". This result occurs because, each day of the Sprint, all members of the Development Team focus on the bigger business picture and align with the ‘why’ of the requirements. Also, this approach helps to horizontally scale the communication of the business intentions when working with several Scrum Teams under one Product Owner and one Product Backlog in a scaled Scrum environment.

ATTD and TDD in Action

The following schematic flow visualizes the steps of driving the development guided by examples in a test driven development fashion. For every User Story from the Product Backlog that has been put into the Sprint Backlog in alignment with the overall Sprint Goal, a set of acceptance criteria are defined. Each of those acceptance criteria, one or more examples are specified. For each example an acceptance test is being written which will fail as the business functionality has not been programmed yet.

The next step is to program the functionality in a TDD way. At some point the test of the example will pass and turn green meaning that all unit tests are passing and the acceptance test is working according to the example. Now is the right time to refactor the code base to keep it clean.

This approach seems to be time intensive at the beginning, especially for developers not well versed in writing tests first. However, in practice it is the basis for a long time sustainable high product development velocity. It allows for fast regression testing which is mandatory when throughput and quality are paramount.

Eylean Scrum Board - Click on ad to reach advertiser web site



PREWISE eylean™

Get more than a Scrum Board!

Replace your regular whiteboard with Eylean board. Use a virtual task board, automatically prepared Burndown charts and enjoy drag and drop interface.

[Learn more](#)

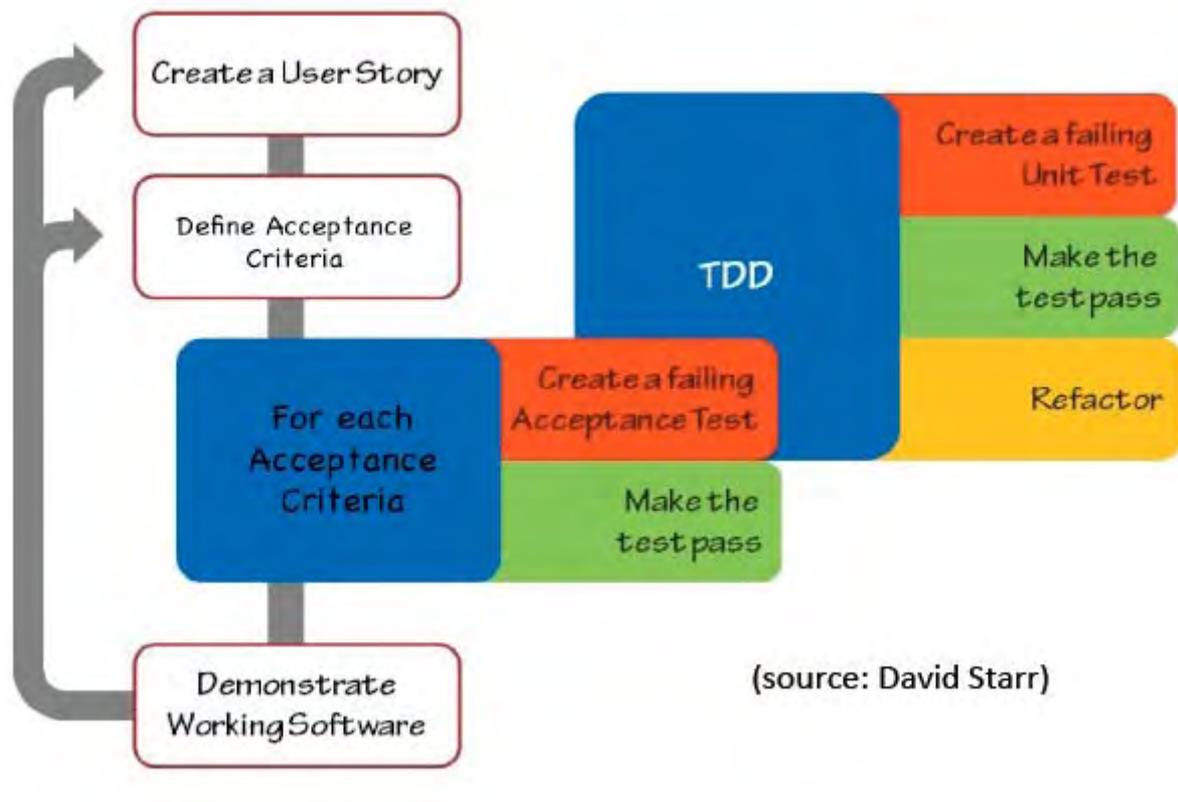


Figure 3 - ATDD to TDD Flow when developing a User Story

Credits

Many thanks to Mario Cardinal of <http://www.mariocardinal.com>. Without his input and support this article wouldn't have been possible.

The details of this article will be proposed by Mario Cardinal and the Ralph Jocham to Scrum.org as an extension to Scrum.

References

1. Scrum Guide, <http://scrum.org/scrumguides>
2. Brian Marrick, <http://www.exampler.com/book/commentary.html>
3. Lisa Crispin, <http://www.methodsandtools.com/archive/archive.php?id=23>
4. Joshua Kerievsky, <http://industriallogic.com/papers/storytest.pdf>
5. Acceptance Test-Driven Development, http://en.wikipedia.org/wiki/Test-driven_development
6. Dan North, <http://dannorth.net/introducing-bdd/>
7. Gojko Adzic, (2011). Specification by Example: How Successful Teams Deliver the Right Software. Greenwich, CT: Manning Publications
8. Robert C. Martin, Grigori Melnik: Tests and Requirements, Requirements and Tests: A Mobius Strip. IEEE Software 25 (1): 54-59 (2008)
9. Ron Jeffries, <http://xprogramming.com/articles/expcardconversationconfirmation/>

Continuous Delivery Using Maven

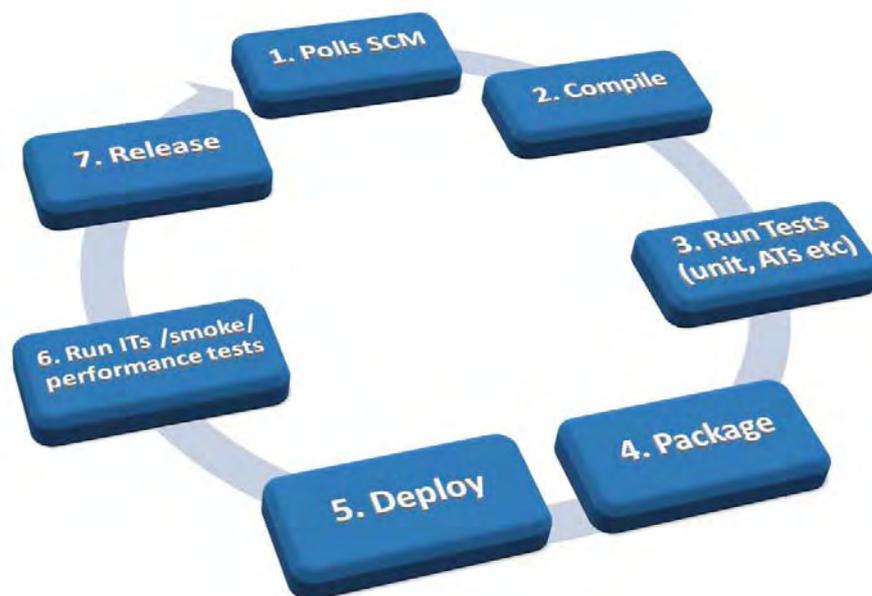
James Betteley, Caplin Systems, <http://jamesbetteley.wordpress.com>

I've recently been working on a Continuous Delivery system using Maven as the build tool. Many people who have used Maven in the past might well consider it to be a poor fit for a Continuous Delivery model, but with a bit of perseverance, I think we've finally got the makings of a pretty good system, which I would like to share with you in this article.

I'll start off with a short introduction to the Continuous Delivery model. Traditional Continuous Integration systems concentrate on running tests and compiling software, occasionally they'll package a build up, and maybe even label it. This is still somewhat short of the package that we want to deliver to production. With continuous delivery, we go a few steps further than traditional C.I. and make each build ready for a deployment to production.

This means that with every check-in, we build our artifacts into a deployable package, include documentation (such as release notes, readme etc), we label each build and store it in a repository ready for deployment, we provide the deploy scripts and we means test the deployment by deploying the build to test environments during the continuous delivery process. The idea is that every build, if it passes all the tests along the way, becomes available for deployment to production.

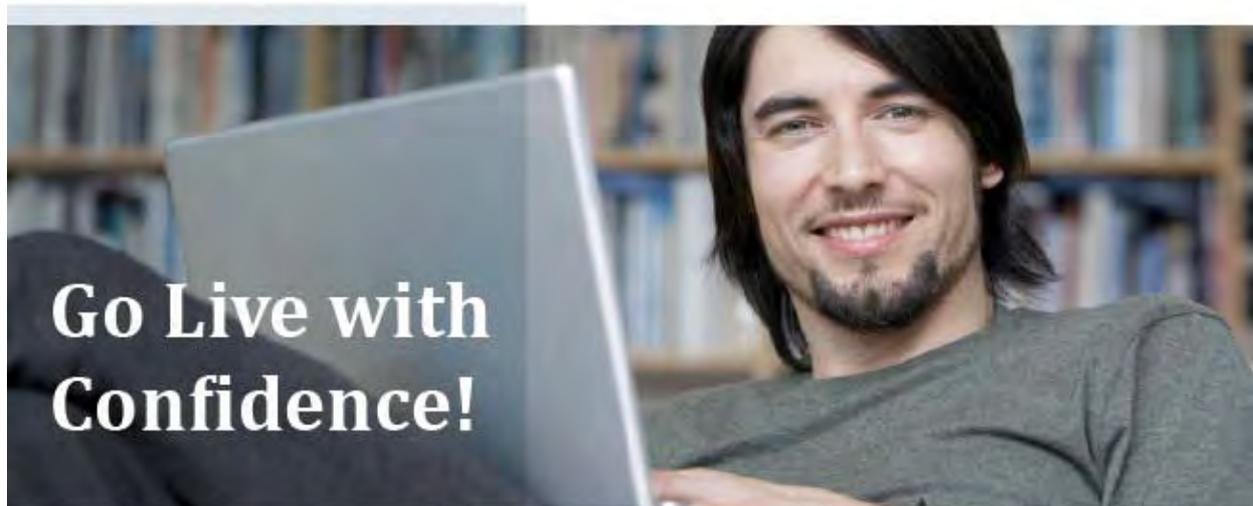
This has numerous advantages over C.I. it means we don't have to do any additional work to our artifacts after they pass testing in order to make them "production ready" and it encourages us (or rather, it forces us) to automate every step along the way, making the whole process more reliable, repeatable and with less risk of human error. The continuous delivery system, in a nutshell, looks a bit like this:



NeoLoad Load and Performance Testing Solution - Click on ad to reach advertiser web site



The Load & Performance Testing Solution for all web & mobile applications



Go Live with Confidence!

“ We were able to identify the system bottleneck before it was a real problem. ”

Chris McCarthy - Terremark

NeoLoad is a load & performance testing solution for web and mobile applications that improves testing effectiveness. It enables faster tests, provides pertinent analysis and supports the newest technologies.

Test your Web application's performance easily and ensure trouble-free deployment thanks to NeoLoad!

Support for HTTP, AJAX, Flex, GWT, Silverlight, Java serialization, Push technologies, Oracle Forms, Siebel...

Generate Virtual Users from:



Your internal infrastructure



The Neotys Cloud Platform

[Download Free Trial](#)

www.neotys.com
E-mail: sales@neotys.com



The Tools

I started out with a bit of a *carte blanche* with regards to what tools to use, but here's a list of what was already in use, in one form or another:

- Ant (the main build tool)
- Maven (used for dependency management)
- CruiseControl
- CruiseControl.Net
- Go
- Monit
- JUnit
- js-test-driver
- Selenium
- Artifactory
- Perforce

The decision of which of these tools to use for my system was influenced by a number of factors. Firstly I'll explain why I decided to use Maven as the build tool.

I'm a big fan of Ant, I'd usually choose it (or possibly even Gradle now) over Maven any day of the week, but there was already an existing Ant build system in place, which had grown a bit monolithic, so I wanted to distance myself from that, and opted for Maven which offers more of a "convention over configuration" approach to build management. I've used Maven before, so I've had my run-ins with it, and I know how hard it can be if you want to do anything outside of "The Maven Way". But the project I was working on seemed pretty simple so Maven got the nod.

GO was the latest and greatest C.I. server in use, and although the enterprise version is pretty expensive, we had a license, so I thought I'd give it a go (no pun intended). Also I'd never used it before so I thought that would be cool, and it's from Thoughtworks Studios, so I thought it might be pretty good. I particularly liked the pipeline feature it has, and the way it manages each of its own agents. I would have opted for Jenkins had there not already been a considerable investment in GO.

Artifactory was chosen as the repository manager, but the system could work just as easily with Sonatype's Nexus, or even with netshares if you didn't want to install a repository manager.

I setup Sonar to act as a build analysis/reporting tool, because we were starting with a Java project. I really like what Sonar does, I think the information it presents can be used very effectively. Most of all I just like the way in which it delivers the information. The Maven site plugin can produce pretty much all of the information that Sonar does, but I think the way Sonar presents the information is far superior – more on this later.

Perforce was the incumbent source control system, and so it was a no-brainer to carry on with that. In fact, changing the SC system wasn't ever in question. That said, I would have chosen Subversion if this was an option, just because it's so utterly free!

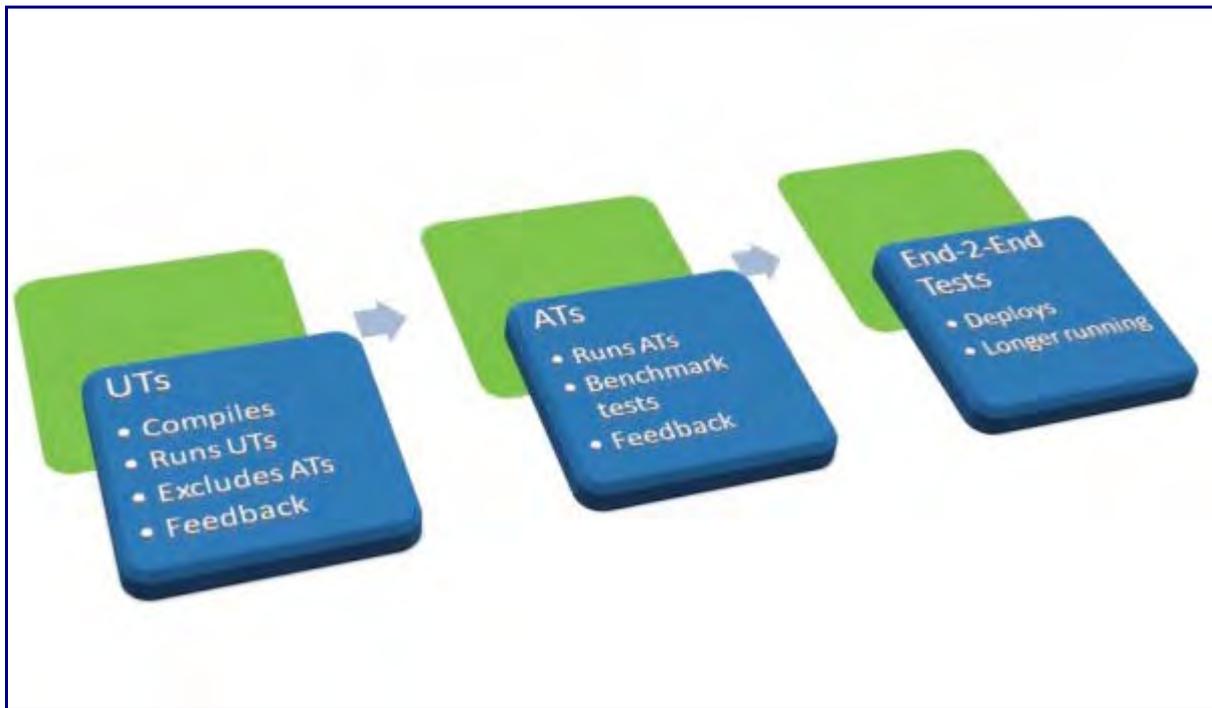
That was about it for the tools I wanted to use. It was up to the rest of the project team to determine which tools to use for testing and developing. All that I needed for the system I was setting up was a distinction between the Unit Tests, Acceptance Tests and Integration Tests. In the end, the team went with Junit, Mockito and a couple of in-house apps to take care of the testing.

The Maven Build, and the Joys of the Release Plugin!

The idea behind my Continuous Delivery system was this:

- Every check-in runs a load of unit tests
- If they pass it runs a load of acceptance tests
- If they pass we run more tests – Integration, scenario and performance tests
- If they all pass we run a bunch of static analysis and produce pretty reports and eventually deploy the candidate to a “Release Candidate” repository where QA and other like-minded people can look at it, prod it, and eventually give it a seal of approval.

This is the basic outline of the build pipeline:



Maven isn't exactly fantastic at fitting in to the pipeline process. For starters we're running multiple test phases, and Maven follows a “lifecycle” process, meaning that every time you call a particular pipeline phase, it runs all the preceding phases again. Our pipeline needs to run the maven Surefire plugin twice, because that's the plugin we use to execute our different tests. The first time we run it, we want to execute all the unit tests. The second time we run it we want to execute the acceptance tests – but we don't want it to run the unit tests again, obviously.

You probably need some familiarity with the maven build lifecycle at this point, because we're going to be binding the Surefire plugin to two different phases of the maven lifecycle so that we can run it twice and have it run different tests each time. Here is the maven default lifecycle:

Telerik Test Studio - Click on ad to reach advertiser web site

Test Studio

Easily record automated tests for your modern HTML5 apps



Test the reliability of your rich, interactive JavaScript apps with just a few clicks. Benefit from built-in translators for the new HTML5 controls, cross-browser support, JavaScript event handling, and codeless test automation of multimedia elements.

 [Download Trial](#)

www.telerik.com/test-studio

Default Lifecycle

- validate
- initialize
- generate-sources
- process-sources
- generate-resources
- process-resources
- compile
- process-classes
- generate-test-sources
- process-test-sources
- generate-test-resources
- process-test-resources
- test-compile
- process-test-classes
- test
- prepare-package
- package
- pre-integration-test
- integration-test
- post-integration-test
- verify
- install
- deploy

Running the Unit Tests

So, we need to bind our Surefire plugin to both the **test** phase to execute the **UTs**, and the **integration-test** phase to run the **ATs**, like this:

```
<plugin>
<!-- Separates the unit tests from the integration tests. -->
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<configuration>
-Xms256m -Xmx512m
<skip>>true</skip>
</configuration>
<executions>
<execution>
<id>unit-tests</id>
<phase>test</phase>
```

```
<goals>
  <goal>test</goal>
</goals>
<configuration>
  <testClassesDirectory>
    target/test-classes
  </testClassesDirectory>
  <skip>>false</skip>
  <includes>
    <include>**/*Test.java</include>
  </includes>
  <excludes>
    <exclude>**/acceptance/*.java</exclude>
    <exclude>**/benchmark/*.java</exclude>
    <include>**/requestResponses/*Test.java</exclude>
  </excludes>
</configuration>
</execution>
<execution>
  <id>acceptance-tests</id>
  <phase>integration-test</phase>
  <goals>
    <goal>test</goal>
  </goals>
  <configuration>
    <testClassesDirectory>
      target/test-classes
    </testClassesDirectory>
    <skip>>false</skip>
    <includes>
      <include>**/acceptance/*.java</include>
      <include>**/benchmark/*.java</include>
      <include>**/requestResponses/*Test.java</exclude>
    </includes>
  </configuration>
</execution>
</executions>
</plugin>
```

Now in the first stage of our pipeline, which polls Perforce for changes, triggers a build and runs the unit tests, we simply call:

```
mvn clean test
```

This will run the surefire test phase of the maven lifecycle. As you can see from the Surefire plugin configuration above, during the “test” phase execution of Surefire (i.e. this time we run it) it’ll run all of the tests except for the acceptance tests – these are explicitly excluded from the execution in the “excludes” section. The other thing we want to do in this phase is quickly check the unit test coverage for our project, and maybe make the build fail if the test coverage is below a certain level.

To do this we use the cobertura plugin, and configure it as follows:

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>cobertura-maven-plugin</artifactId>
  <version>2.4</version>
  <configuration>
    <instrumentation>
      <excludes>
        <exclude>*/acceptance/*.class</exclude>
        <exclude>*/benchmark/*.class</exclude>
        <exclude>*/requestResponses/*.class</exclude>
      </excludes>
    </instrumentation>
    <check>
      <haltOnFailure>>true</haltOnFailure>
      <branchRate>80</branchRate>
      <lineRate>80</lineRate>
      <packageLineRate>80</packageLineRate>
      <packageBranchRate>80</packageBranchRate>
      <totalBranchRate>80</totalBranchRate>
      <totalLineRate>80</totalLineRate>
    </check>
    <formats>
      <format>html</format>
      <format>xml</format>
    </formats>
  </configuration>
  <executions>
    <execution>
      <phase>test</phase>
      <goals>
        <goal>clean</goal>
        <goal>check</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

I changed the lifecycle phase that the cobertura plugin binds to, to stop it from running the integration-test phase all over again. I've made it bind to the test phase only, so that it only executes when the unit tests run. A consequence of this is that we can now change the maven command we run, to something like this:

```
mvn clean cobertura:cobertura
```

This will run the Unit Tests implicitly and also check the coverage!

Executing the Acceptance Tests

In the second stage of the pipeline, which runs the acceptance tests, we can call:

```
mvn clean integration-test
```

Don't invest in multiple automated testing tools when all you need is QA Wizard Pro!



You can do it all with **QA Wizard Pro:**

- Web, Windows, and Java testing
- Functional, stress, and load testing
- Object-based recording and intelligent scripting

Automated testing is a large investment in time and resources. Make sure your tool supports the technologies and platforms you use today, and will grow with you in the future.

Visit www.seapine.com/onetool
to get your free **QA Wizard Pro** trial now!

This will again run the Surefire plugin, but this time it will run through the test phase (thus executing the unit tests again) and then execute the integration-test phase, which actually runs our acceptance tests.

You'll notice that we've run the unit tests twice now, and this is a problem. Or is it? Well actually no it isn't, not for me anyway. One of the reasons why the pipeline is broken down into sections is to allow us to separate different tasks according to their purpose. My Unit Tests are meant to run very quickly (less than 3 minutes ideally, they actually take 15 seconds on this particular project) so that if they fail, I know about it asap, and I don't have to wait around for a lifetime before I can either continue checking in, or start fixing the failed tests. So my unit test pipeline phase needs to be quick, but what difference does an extra few seconds mean for my Acceptance Tests? Not too much to be honest, so I'm actually not too fussed about the unit tests running for a second time. If it was a problem, I would of course have to skip the unit tests, but only in the test phase on the second run. This is doable, by using the failsafe plugin.

Deploying to Artifactory

The next thing we want to do is create a built artifact (a jar or zip for example) and upload it to our artifact repository. We'll use 5 artifact repositories in our continuous delivery system, these are:

1. A cached copy of the maven central repo
2. A C.I. repository where all builds go
3. A Release Candidate (RC) repository where all builds under QA go
4. A Release repository where all builds which have passed QA go
5. A Downloads repository, from where the downloads to customers are actually served

Once our build has passed all the automated test phases it gets deployed to the C.I. repository. This is done by configuring the C.I. repository in the maven pom file as follows:

```
<distributionManagement>
  <repository>
    <id>CI-repo</id>
    <url>http://artifactory.mycompany.com/ci-repo</url>
  </repository>
</distributionManagement>
```

and calling:

```
mvn clean deploy
```

Now, since Maven follows the lifecycle pattern, it'll rerun the tests again, and we don't want to do all that, we just want to deploy the artifacts. In fact, there's no reason why we shouldn't just deploy the artifact straight after the Acceptance Test stage is completed, so that's what exactly what we'll do. This means we need to go back and change our maven command for our Acceptance Test stage as follows:

```
mvn clean deploy
```

This does the same as it did before, because the integration-test phase is implicit and is executed on the way to reaching the "deploy" phase as part of the maven lifecycle, but of course it does more than it did before, it actually deploys the artifact to the C.I. repository.

Why I Don't Use the Release Plugin

One thing that is worth noting here is that I'm not using the maven release plugin, and that's because it's not very well suited to continuous delivery. The main problem is that the release plugin will increment the build number in the pom and check it in, which will in turn kick off another build, and if every build is doing this, then you'll have an infinitely building loop.

Maven declares builds as either a "release build" which uses the release plugin, or a SNAPSHOT build, which is basically anything else. But I want to create releases out of SNAPSHOT builds, but I don't want them to be called SNAPSHOT builds, because they're releases! So what I need to do is simply remove the word SNAPSHOT from my pom. Get rid of it entirely. This will now build a normal "snapshot" build, but not add the SNAPSHOT label, and since we're not running the release plugin, that's fine.

Warning: if you try removing the word snapshot from your pom and then try to run a release build using the release plugin, it'll fail.

Ok, let's briefly catch up with what our system can do now:

- We've got a build pipeline with 2 stages
- It's executed every time code is checked-in
- Unit tests are executed in the first stage
- Code coverage is checked, also in the first stage
- The second stage runs the acceptance tests
- The jar/zip is built and deployed to our C.I. repository, this also in the second stage of our pipeline

So we have a jar, and it's in our C.I. repository, and we have a code coverage report. But where's the rest of our static analysis? The build should report a lot more than just the code coverage. What about coding styles and standards, rules violations, potential defect hot spots, copy and pasted code etc and so forth??? Thankfully, there's a great tool which collects all this information for us, and it is called Sonar.

Sonar

Once you've installed Sonar somewhere (which is exceedingly easy), getting your builds to produce Sonar reports is as simple as adding a small amount of configuration to your pom, and adding the Sonar plugin to your plugin section. To produce the Sonar reports for your project, you can simply run:

```
mvn sonar:sonar
```

So that's exactly what we'll do in the next section of our build pipeline.

So we now have 3 pipeline sections and we're producing Sonar reports with every build. The Sonar reports look something like this:



As you can see, Sonar produces a wealth of useful information which we can pour over and discuss in our daily stand-ups. As a rule, try to fix any “critical” rule violations, and keep the unit test coverage percentage up in the 90s (where appropriate). Some people might argue that unit test coverage isn't a valuable metric, but bear in mind that Sonar allows you to exclude certain files and directories from your analysis, so that you're only measuring the unit test coverage of the code you *want* to have covered by unit tests. For me, this makes it a useful metric.

Executing the Integration Tests

Moving on from Sonar now, we get to the next stage of the pipeline, and here I'm going to run some Integration Tests (finally!). The Integration Tests have a much wider scope than the Unit Test, and they also have greater requirements, in that we need an Integration Test Environment to run them in. I'm going to use Ant to control this phase of the pipeline, because it gives me more control than Maven does, and I need to do a couple of funky things, namely:

- Provision an environment
- Deploy all the components I need to test with
- Get my newly built artifact from the C.I. repository in Artifactory
- Deploy it to my test environment
- Kick off the tests

The Ant script is fairly straightforward, but I'll just mention that getting our artifact from Artifactory is as simple as using Ant's own “get” task (you don't need to use Ivy juts to do this):

```
<get src="${artifactory.url}/${repo.name}/${namespace}/${jarname}-${version}"  
dest="${temp.dir}/${jarname}-${version}" />
```

The Integration Test stage takes a little longer than the previous stages, and so to speed things up we can run this stage in parallel with the previous stage. Go allows us to do this by setting up 2 jobs in one pipeline stage, with jobs running in parallel. The Jenkins pipeline plugin has the same functionality.

Once this phase completes successfully, we know we've got a decent build! At this point I'm going to throw a bit of a spanner into the works. The QA team wants to perform some manual exploratory tests on the build. Good idea! But how does that fit in with our Continuous Delivery

model? Well, what I did was to create a separate “Release Candidate” (RC) repository, also known as a QA repository. Builds that pass the IT stage get promoted to the RC repository, and from there the QA team can take them and do their exploratory testing.

Does this stop us from practicing “Continuous Delivery”? Well, not really. In my opinion, Continuous Delivery is more about making sure that every build creates a *potentially releasable artifact*, rather than making every build actually deploy an artifact to production – that’s Continuous Deployment.

Our final stage in the deployment pipeline is to deploy our build to a performance test environment, and execute some load tests. Once this stage completes we deploy our build to the Release Repository, as it’s all signed off and ready to handover to customers. At this point there’s a manual decision gate, which in reality is a button in my CI system. At this point, only the product owner or some such responsible person can decide whether or not to actually release this build into the wild.

They may decide not to, simply because they don’t feel that the changes included in this build are particularly worth deploying. On the other hand, they may decide to release it, and to do this they simply click the button. What does the button do? Well, it simply copies the build to the “downloads” repository, from where a link is served and sent to customers, informing them that a new release is available – that’s just one way of doing it. In a hosted environment (like a web-based company), this button-press could initiate the deploy script to deploy this build to the production environment.

A Word on Version Numbers

This system is actually dependent on each build producing a **unique** artifact. If a code change is checked in, the resultant build must be uniquely identifiable, so that when we come to release it, we know we’re releasing *the exact same build that has gone through the whole pipeline*, not some older previous build. To do this, we need to version each build with a unique number. The CI system is very useful for doing this. In Go, as with most other CI systems, you can retrieve a unique “counter” for your build, which is incremented every time there’s a build. No two builds of the same name can have the same counter. So we could add this unique number to our artifact’s version, something like this (let’s say the counter is 33, meaning this is the 33rd build):
myproject.jar-1.0.33

This is good, but it doesn’t tell us much, apart from that this is the 33rd build of “myproject”. A more meaningful version number is the source control revision number, which relates to the code commit which kicked off the build. This is extremely useful. From this we can cross reference every build to the code in our source control system, and this saves us from having to “tag” the source code with every build.

I can access the source control revision number via my CI system, because Go sets it as an environment variable at build time, so I simply pass it to my build script in my CI system’s xml, like this:

```
mvn cobertura:cobertura -Dp4.revision=${env.GO_PIPELINE_LABEL}  
-Dbuild.counter=${env.GO_PIPELINE_COUNTER}
```

p4.revision and build.counter are used in the maven build script, where I set the version number:

```
<groupId>com.mycompany</groupId>
<artifactId>myproject</artifactId>
<packaging>jar</packaging>
<version>${main.version}-${build.number}-${build.counter}</version>
<name>myproject</name>
<properties>
<build.number>${p4.revision}</build.number>
<major.version>1</major.version>
<minor.version>0</minor.version>
<patch.version>0</patch.version>
<main.version>${major.version}.${minor.version}.${patch.version}</main.version>
</properties>
```

If my Perforce check-in number was 1234, then this build, for example, will produce:

```
myproject.jar-1.0.0-1234-33
```

And that just about covers it. I hope this is useful to some people, especially those who are using Maven and are struggling with the release plugin!

DSDM Atern Overview

Matthew Caine

M.C. Partners & Associates, <http://www.mcpa.biz/>

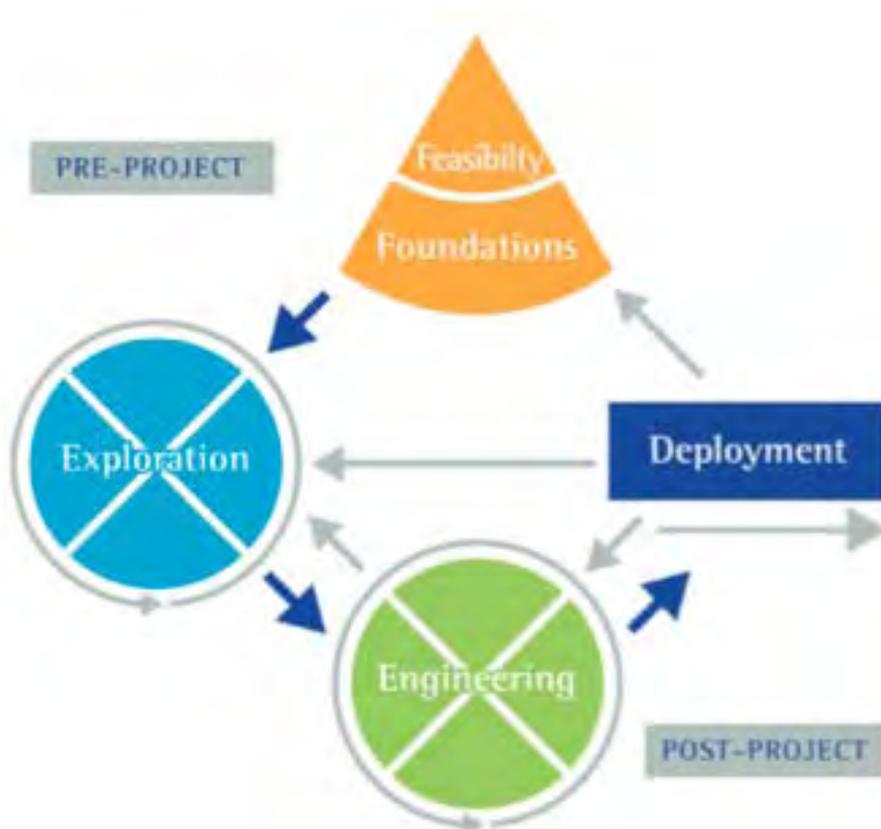
DSDM (Dynamic Systems Development Method), the longest-established Agile method, launched in 1995, is the only Agile method to focus on the management of Agile projects. Arie van Bennekum represented DSDM at the launch of the Agile Alliance and their Agile Manifesto in 2001. DSDM has mainly operated in the corporate environment where it consistently demonstrates its ability to successfully work within and complement existing corporate processes. Practicing evolutionary development itself DSDM's latest version (Atern) incorporates those improvements.

This article provides a high-level introduction to Atern: its structure and phases, principles, roles and responsibilities and a brief look at the products.

The Structure of an Atern Project

Atern differs from more common agile approaches as it encompasses the entire project lifecycle and not just software development (where Scrum prevails). It incorporates project management disciplines and provides mechanisms to ensure that the project benefits are clear, the proposed solution is feasible and there are solid foundations in place before detailed work is started.

There are seven phases to an Atern project:



Jama Contour Collaborative Requirements Management - Click on ad to reach advertiser web site



Jama Contour

Collaborative requirements management.

Leverage your collective genius. Deliver successful projects.

At Jama, we believe collaboration is the key to success. Teams developing complex systems, software and other products use Contour, the powerful Web-based requirements management software, to manage the detailed scope of projects through the development lifecycle. People love Contour because it keeps everyone connected, fits any process and is elegantly simple to use.

FREE TRIAL

Try Contour free for 30 days.

No installation needed. Sign up now for your Contour hosted trial.

www.jamasoftware.com
Email us: info@jamasoftware.com
Call us: 1 (800) 679-3058

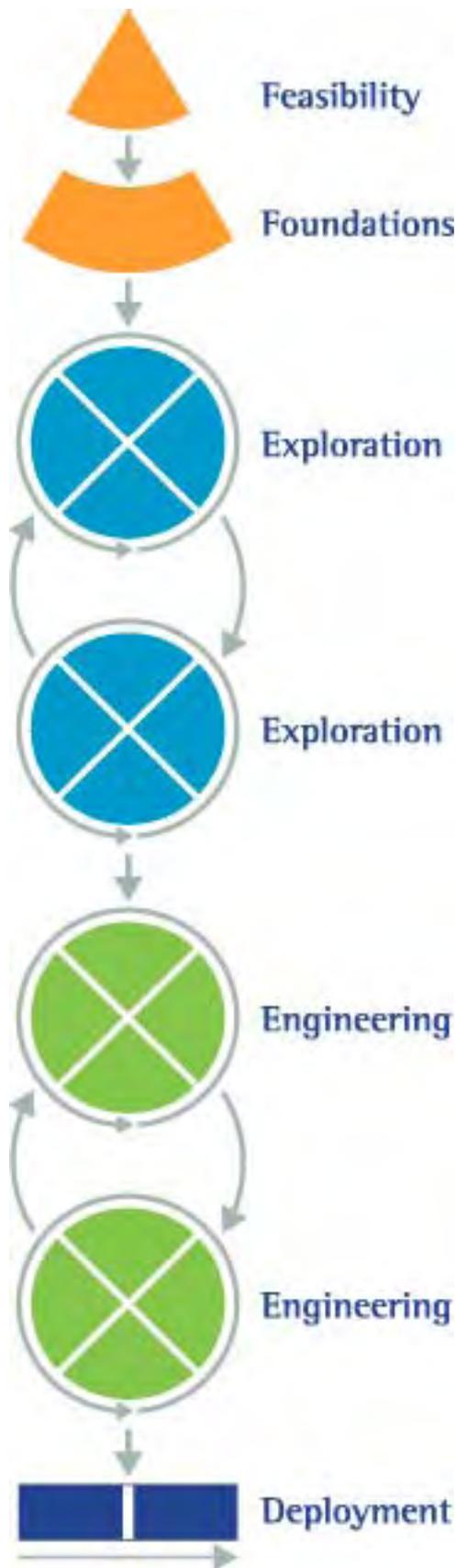


Phase	Key Responsibilities
Pre-project	Initiation of the project, agreeing the Terms of Reference for the work
Feasibility	Typically a short phase to assess the viability and the outline business case (justification).
Foundations	Key phase for ensuring the project is understood and defined well enough so that the scope can be baselined at a high level and the technology components and standards agreed, before the development activity begins.
Exploration	Iterative development phase during which teams expand on the high level requirements to demonstrate the functionality
Engineering	Iterative development phase where the solution is engineered to be deployable for release
Deployment	For each Increment (set of timeboxes) of the project the solution is made available.
Post project	Assesses the accrued benefits.

The Exploration and Engineering phases are often merged, as the method is flexible, allowing them to be organized to best suit the situation. Some examples are provided below:



Example 1 illustrates iterative development with the solution evolving over a number of Exploration- Engineering cycles before Deployment of an increment.



Example 2 completes all Exploration activities prior to commencing the Engineering activities. The timeboxed Iterative Development occurs within the stage as opposed to the previous and following example. This approach is not to be confused with a traditional waterfall approach.

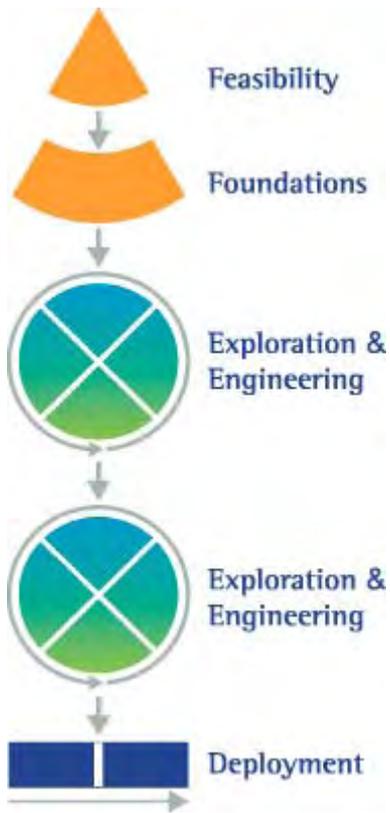


AGILE2012
DALLAS, TEXAS **AUG 13-17**

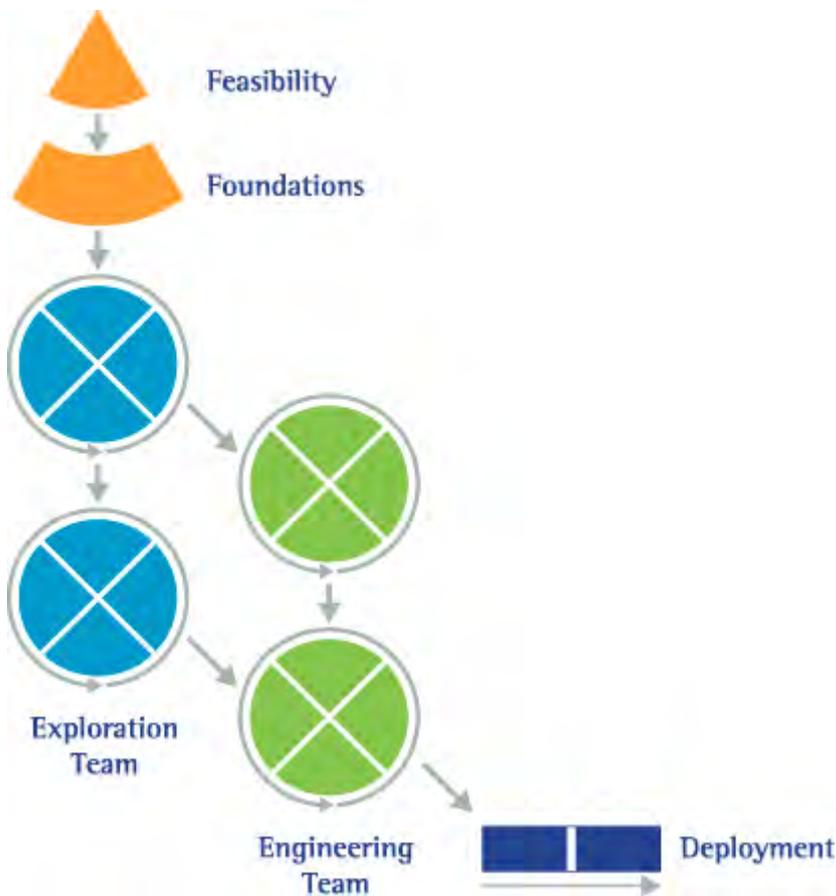
<http://agile2012.agilealliance.org>

PRESENTED BY 





Example 3 combines Exploration and Engineering work to deliver fully engineering subsets of the end product in a single pass.



Example 4 reflects a more complex scenario with two teams involved. For simplicity, two teams are shown but, in practice, several teams could be involved if the size and complexity requires them. One team concentrates on exploratory work and the other on engineering. In this example, the Exploration team might deliver prototypes of the solution to the Engineering team who then build solutions for Deployment.

Atern Principles

Many organisations guide general behaviour with high-level values and culture. Well-understood principles are better guides than detailed process procedures. In Atern principles are used to provide guidance throughout the project.

Atern has eight underlying principles and the complete framework can be directly derived from these. The principles are based on best practice in its truest sense. They define “the way things are done”.

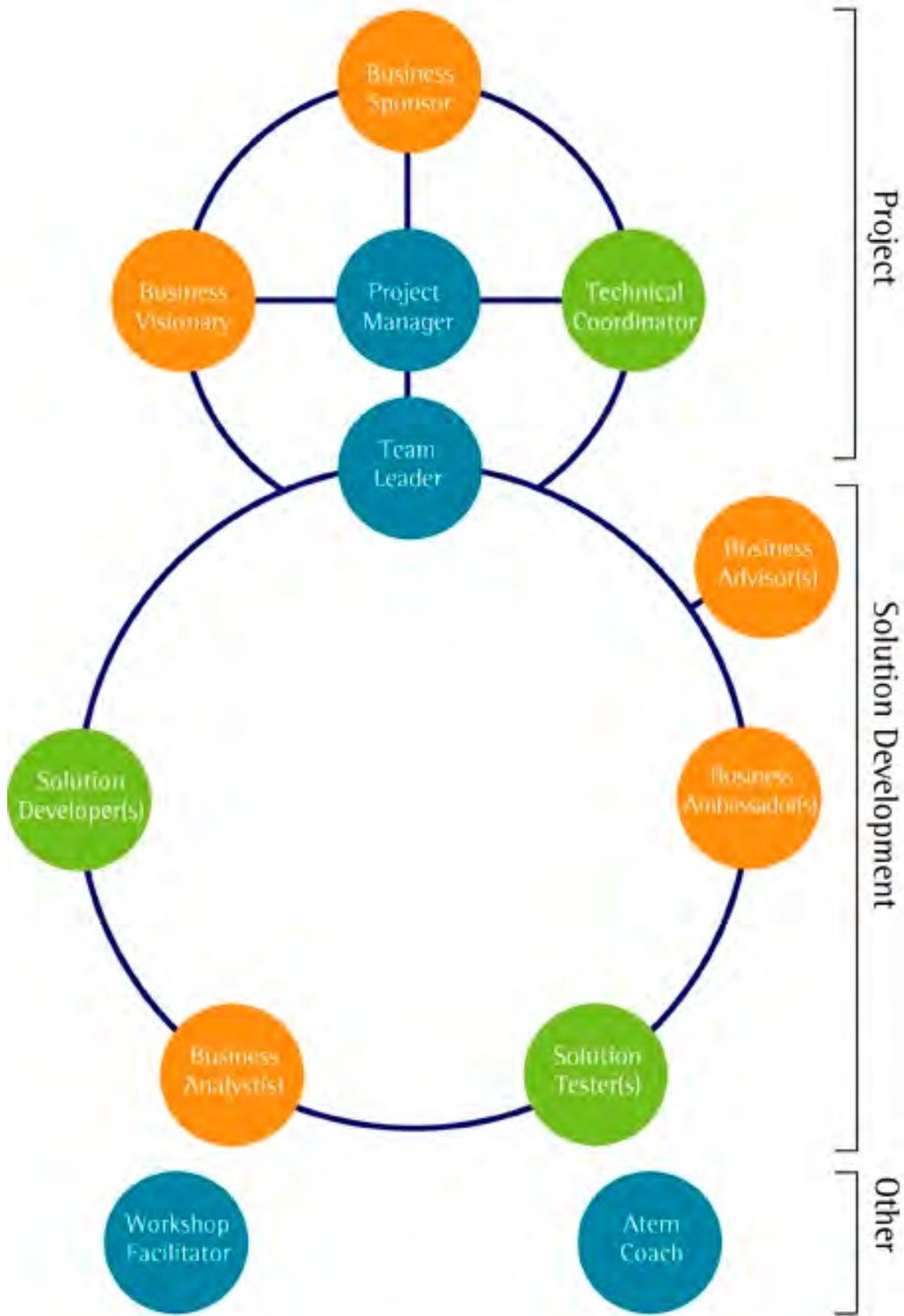
Breaking one of these principles can lead to failure, as these are the basic building blocks for Atern, and bind together all the other elements of Atern.

Principal	Description
Focus on the Business Need	Deliver what the business needs when it needs it. The true business priorities must be understood with a sound business case.
Deliver on Time	Timeboxes are planned in advance and the timeframe set. The dates never change; features are varied depending on business priorities, in order to achieve the deadline.
Collaborate	Teams work in a spirit of active co-operation and commitment. Collaboration encourages understanding, speed and shared ownership. The teams must be empowered and include the business representatives.
Never Compromise on Quality	A solution has to be “good enough”. The level of quality is set at the outset. Projects must test early and continuously and review constantly.
Build Incrementally from Firm Foundations	Increments allow the business to take advantage of work before the final product is complete, encouraging stakeholder confidence and feedback. This is based on doing just enough upfront analysis to proceed and accepting that detail emerges later.
Develop Iteratively	Accept that work is not always right first time. Use Timeboxes to allow for change yet continuously confirm that the solution is the right one.
Communicate Continuously and Clearly	Use facilitated workshops, daily standups, modeling, prototyping, presentations and encourage informal face-to-face communication.
Demonstrate Control	The team needs to be proactive when monitoring and controlling progress in line with Foundations Phase. They need to constantly evaluate the project viability based on the business objectives.

The Roles and Responsibilities of an Atern Project

Atern defines the roles and responsibilities in such a way that it easy to imagine how existing roles and positions would fit into an Atern project.

Atern Roles



© 2008 Dynamic Systems Development Method Limited

Descriptions for each role are described on the next page.

Project Roles

Role	Key Responsibilities
Business Sponsor	Owns the business case. Ensures funding and resourcing. Guarantees effective decision-making and deals with escalations rapidly.
Project Manager	Entry point for project governance. High-level planning. Monitors progress, resource availability, project configuration, manages risk and escalated issues.
Business Visionary	Owns the business vision and impact on wider business changes. Monitors progress against the vision. Contributes to key requirements, design and review sessions.
Technical Coordinator	Agrees and controls technical architecture. Advises and co-ordinates teams. Identifies and manages technical risk. Ensures non-functional requirements are met.

Solution Development Roles

Role	Key Responsibilities
Team Leader	Focuses team to deliver on time. Encourages full team participation. Manages detailed time box activities and day-to-day activities. Ensures testing and review activities are scheduled and completed.
Business Ambassador	Contributes to all requirements, design and review sessions. Provides the business view for all day-to-day decision making. Describes business scenarios to help design and test the solution. Provides assurance that the solution is correct. Coordinates business acceptance.
Solution Developer	Creates the solution and participates fully in all appropriate QA activities.
Solution Tester	Works with business roles to define test scenarios for the solution. Carries out full technical testing reporting results to the Team Leader and Technical Coordinator.
Business Analyst	Supports communication between business and technical members of the team. Manages all required products related to business requirements. Ensures business implications of day-to-day decisions are properly thought through.
Business Advisor	Provides specialist input, for example an accountant or a tax advisor. Usually an intended user of the solution.

Other Roles

Role	Key Responsibilities
Atern Coach	Helps teams new to Atern teams get the most out of Atern. Tailors Atern for the needs of the project. Not all aspects are needed all the time!
Workshop Facilitator	Manages and organizes workshops. Responsible for the context not the content. Independent.
Other Specialists	Experts required on a short-term basis, possibly technical e.g. Load-Test specialists etc.

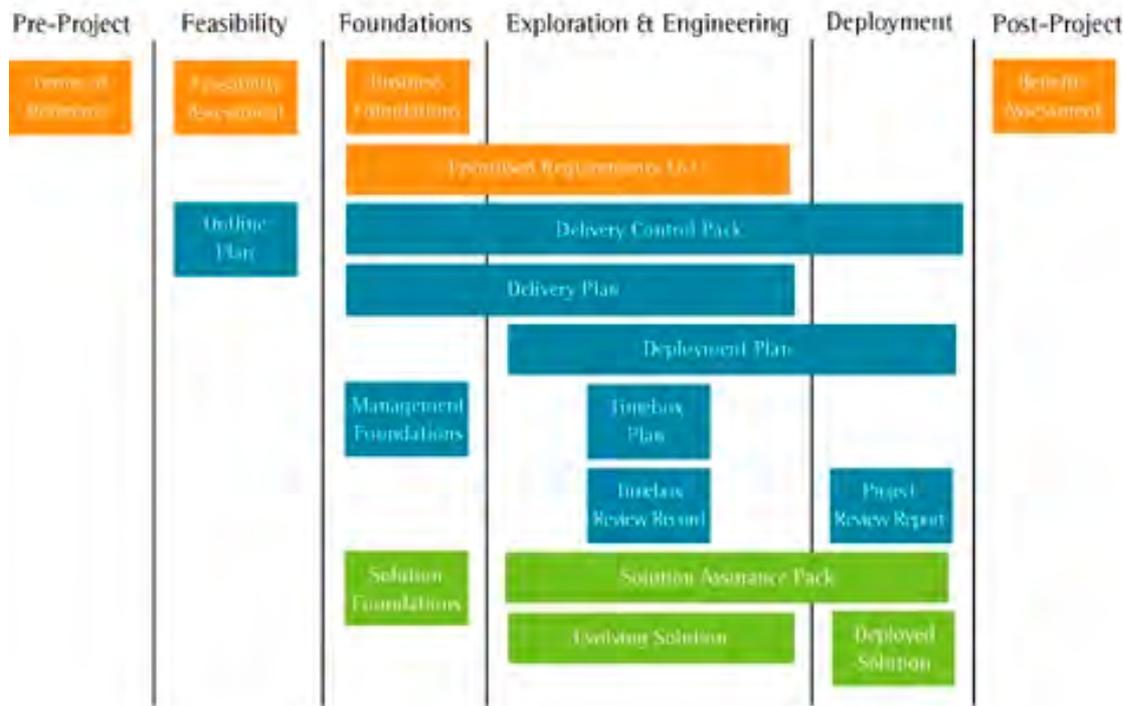
The Atern Products

Deliverables are associated with each phase of the lifecycle. These are referred to as products. Not all products are required for every project and the formality will vary according to the project and organisation. Influencing factors could be contractual relationships and corporate standards.

Some products are specific to a particular phase in the lifecycle, others may continue to evolve through subsequent phases.

The basic flow of products through the lifecycle is shown below. For instance, the Feasibility Assessment enhanced with Business Foundations and the Prioritised Requirements List (PRL). Similarly, the Outline Plan is refined into the Delivery Plan for the project that in turn the teams refine to create the individual Timebox Plans and the Deployment Plan for an increment.

Atern Products



© 2008 Dynamic Systems Development Method Limited

Atern allows the project to decide for itself how the products are built or what they should look like, allowing products to be tailored to most environments. Indeed, some environments will require all products and others only the PRL and Evolving Solution (similar to Scrum).

Knowledge Management and Software Organizations

Bhaskar Raju, bhaskar_raju [at] xyratex.com
Xyratex Technology Limited, <http://www.xyratex.com/>

*Is it hard to get expertise in the work place for employees?
Are you not able to handle the ever changing requirements and improve the quality?
Are you not able to manage teams effectively and improve employee productivity?
Is the 'skills gap' widening in your teams?*

Organizations are looking for better management options, to address the above questions. To deal with the issues in this truly competitive and fast-paced business environment, it is essential for organizations to recognize the value of knowledge and manage the knowledge assets. Many organizations don't realize how much Knowledge Management (KM) contributes to phenomenal growth of organizations by transforming new as well as existing enterprise knowledge into superior products/services/solutions. This article emphasizes the significance of KM practices in the context of the software industry in resolving above issues. It will also covers how knowledge conversion takes place in organizations, why organizations fail to utilize Knowledge Management in solving the business problems in long-term and how KM can be implemented effectively with best practices.

Changing nature of customer needs / requirements

Frequently changing customer requirements and high levels of customer expectations exist in almost all the domains of software industry. This became challenging for various teams in these software organizations that have to balance autonomy with the need to respond to external forces and inter-team responsibilities. In this context, various skills, capabilities or competencies which make up software team's know-how factors are considered to be the principle resources of organizations [1]. Managing the team's knowledge plays an important role in developing these skills, capabilities or competencies for successful release of products/services to the customers.

Understanding knowledge management

Knowledge management is the process that involves a cycle of exchanging tacit knowledge and converting it to explicit knowledge and then re-formulating it through an individual's experience and other factors (such as belief, perspective, and values) into tacit knowledge [2].

Adminitrack Issue & Defect Tracking - Click on ad to reach advertiser web site

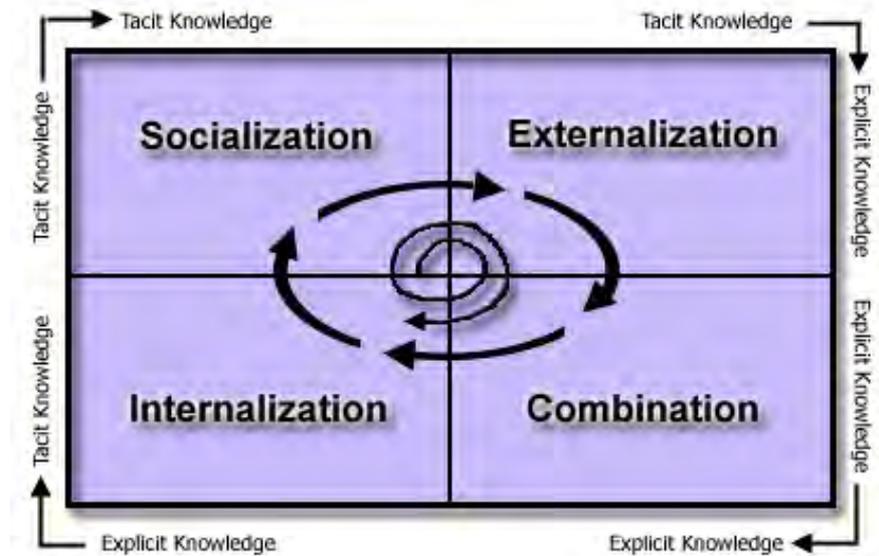


Issue & Defect Tracking
Most Effective Solution for Professional Teams

“Got an eye on all of your project team's issues?”

Free 30-Day Trial Now Available at
www.AdminiTrack.com

According to literature, there are two main types of knowledge: tacit and explicit. Tacit knowledge refers to personal knowledge embedded in individual experience and involving intangible factors. This type of knowledge can be considered to be very difficult to transfer. On the other hand, explicit knowledge refers to the one that has already been documented and articulated into formal language, and can be much more easily accessible and transferred among individuals. Hence, one of the key functions of a KM strategy is to make tacit knowledge explicit.



The success of an organization depends on how well, it converts tacit knowledge into explicit. According to Nonaka, who is best known for his study on Knowledge management, there are 4 types of knowledge conversion between tacit and explicit knowledge [3]. According to his model of knowledge creation and transformation, tacit knowledge is exchanged with tacit knowledge through socialization. Tacit knowledge can be converted to explicit knowledge through externalization where the hidden know-how is expressed and articulated through metaphors, models, concepts, equations and other forms of explanation. Explicit knowledge can be exchanged and developed through communication. Explicit knowledge is converted into tacit knowledge through internalization where individuals absorb it through experience, testing and/or simulating their use of operational knowledge. Hence, these four different types of conversion can promote the generation of important intangible knowledge assets which contribute to long lasting competitive advantage.

KM to solve software organization problems

We describe below various challenges that software organizations are facing in the process of meeting the high customer expectation levels and how KM practices can help them to face these challenges.

Global distributed teams: An increasing number of software organizations are relying on technology-enabled geographically distributed teams. For example, mobile phone software development teams in various organizations spread across different locations employs experienced developers in different countries to solve their technical challenges and develop new features. In this context, managing teams effectively at different locations is a challenging task for any organization. Improving productivity and driving them towards the same goal requires readily sharing information across sites. Formal KM systems can facilitate this.

By enabling efficient sharing mechanisms to developers, KM plays important role if organization is also involved in Open Source Software projects, as the quality of the deliverable depend on how well the parallel development on independent software components across global teams are synchronized.

Iterative-incremental development: Due to the increasing demand by customers to change requirements very frequently, iterative-incremental development approaches are becoming popular in many of software organizations. On average, projects can be divided into 12 one- to four-week iterations. As development involves successive iterations, the required time can be reduced based on the experience gained from past iterations. Unless the knowledge developed in each iteration is managed efficiently and change control mechanisms are in place, it becomes very complex for organizations to deliver project without delays and cost overruns. KM defines a standard process, so that engineers will record the related knowledge in each iteration into knowledge based databases and systemizes knowledge acquisition process. By making knowledge acquisition processes continuous, KM combines the knowledge developed on requirements, functionality, design, coding, testing in earlier iterations and reconnects it to the required knowledge for implementation in future iterations.

Quick testing life cycles: During the last phase of quick software development life cycles, test teams aimed to capture defects by integrating software, hardware and computer systems. Nowadays, test life cycle times are becoming shorter and shorter due to the need for quick delivery to customers. This situation demands good knowledge on the overall system, supported features, and strong analysis as well as decision making capabilities to ensure defects do not escape in the software releases. To complete test cycles quickly, organizations are looking for various automation testing methods which are reliable in terms of reusability and productivity compared to manual testing. Even though testing is executed automatically, test teams require knowledge of various tested features and scripts, and analytical capabilities to isolate software issues in the system. With experience in problem solving well-defined in knowledge based database, test engineers can always search for related issues raised in past cycles and reuse the solution [4]. Besides that, KM systems provide a search and consult mechanism to support test engineers in information search and decision making. With less effort spent on unnecessary issue investigation combined with effective analysis, test cycle will be completed more quickly thus contributing to test efficiency.

Merger and acquisitions: Today's trends towards mergers and acquisitions increases the challenge of integration. This can pose major threat to organizations that might struggle to take certain organizational decisions, fearing to loose knowledge by reducing the teams. Once the KM system has achieved a matured level, the system may be enhanced or evolved to a more intelligent system that is capable of decision making. If the knowledge-based rational decisions are not taken, people critical to core competencies may walk away and put the organizations at risk. Successful organizations have a key part of their success in their expertise to manage knowledge systems.

Small-medium organizations: No matter at what stage of development life cycle we are at and no matter the size of organization, KM practices are significant. Let's look at the example of small and medium sized software organizations in data storage domain. Given the growth opportunity for the data storage markets, various small and medium sized organizations design and develop a range of advanced, scalable data storage solutions for the Original Equipment Manufacturer (OEM) community. OEMs are highly dependent on organizations which provide enterprise-class data storage subsystems. The software divisions in these small/medium organizations are experts in delivering bespoke software solutions to their customers based on their requirement. Different sub-teams within the same software group develop and cut the

release branches quickly and with top-notch quality designed to meet every client's needs. The question arises on how to handle the ever changing requirements and improve the quality of code. Even though the bespoke software solutions developed on the same trunk of code, it's very important for these teams to share best practices and relevant differential knowledge across sub-teams. KM encourages team members to record significant changes during bespoke software development and this will be shared across different sub-teams to re-integrate this learning. This gives opportunity to better exploit existing knowledge assets by re-deploying them to their sub-teams, modifying knowledge from a past process to create a new solution.

Another challenge for these organizations to manage multiple sub-teams by moving the team members depending on the workload to meet growing customer expectation levels. In these small/medium organizations, with lack of consistent processes across the organization, most of projects will be dominated by few more experienced team members. If the efficient KM systems are not placed in organizations where individual resources play a critical role, team members will find it hard to get the expertise on the overall systems. By setting up a mentoring relationship between experienced experts and new members, implementing a document management system to provide access to key explicit knowledge, KM tries to tap into experts' knowledge banks and share their know-how with others.

Software outsourcing models: Due to the popularity of software outsourcing models, organizations are moving their work to lower cost destinations such as India and the Far East to gain cost advantage. This has posed its own set of challenges as well as opportunities, since countries like India with a well-educated and experienced workforce also have severe talent shortages. Organizations in these countries are developing comprehensive and forward-looking strategies to recruit, develop, and retain the best professionals. As attrition rates are very high due to the high demand of skill sets, organizations must concentrate on an efficient cycle of exchanging tacit knowledge and converting it to explicit knowledge. With experienced members leaving the team and with increasing numbers of new recruits, skill gaps widen within the teams. KM practices provide effective tools for smooth knowledge transfer to new team members and give the opportunity to build competencies by narrowing the skills gap.

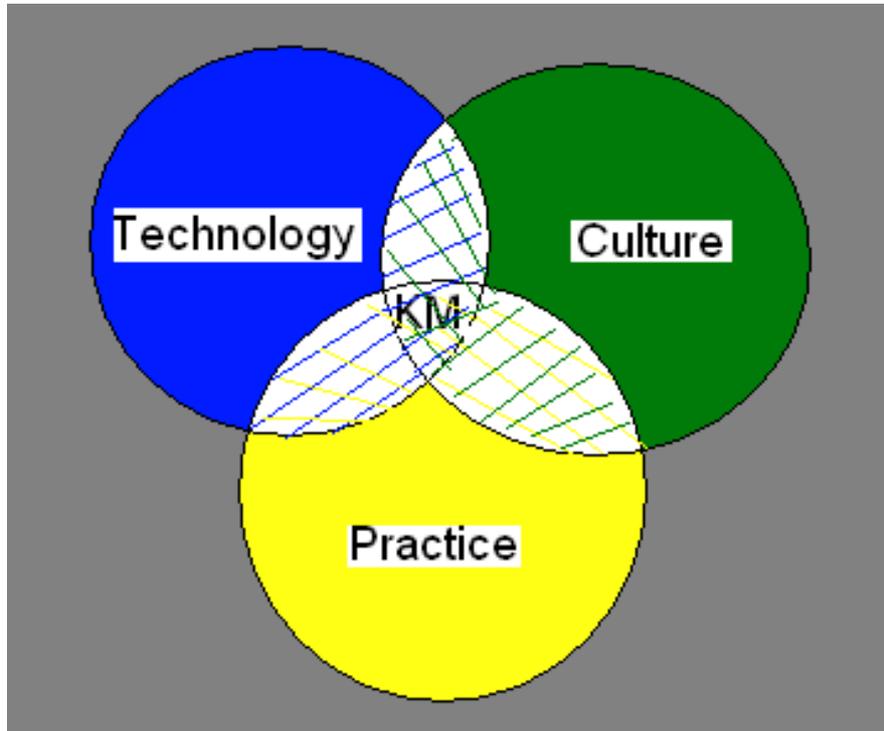
Why do organizations fail to utilize KM?

Technology, people, practice

Despite the fact that KM adds so much value to the organizations in long-term, most of organizations fail to utilize KM in solving the business problems. At the initial stage, organizations may try to align KM with the overall business objectives, but they often fail to retain these practices over time. The reason for this is that organizations fail over a period of time to distinguish between applications that manage explicit knowledge and the whole KM process.

Management has to invest a lot of money in KM systems and spend a lot of time managing the applications such as databases or searchable repositories. Even while reviewing KM projects, they emphasis more on KM systems and cataloguing existing explicit information. For successful KM, this is not enough. There must be efficient cycles of exchanging tacit knowledge and converting it to explicit knowledge and then re-formulate it through an individual's experience and other factors (such as belief, perspective, and values) into tacit knowledge. By practicing these cycles of conversion and updating the system regularly with explicit knowledge during this conversion process, teams can benefit from KM activities.

For instance, in software organizations, testing teams put a significant amount of effort into validation and verification of software release by the development teams. In some cases, efforts spent by the developers will result in defects being closed as duplicate or "as designed". These organizations will have efficient systems to log test information, but they forget to account for certain critical elements which enable knowledge sharing between their employees. We must agree that knowledge dies when it is disembodied [5]. By promoting a knowledge sharing culture within the teams, management must reduce the team's efforts spent in reinventing the wheel activities.

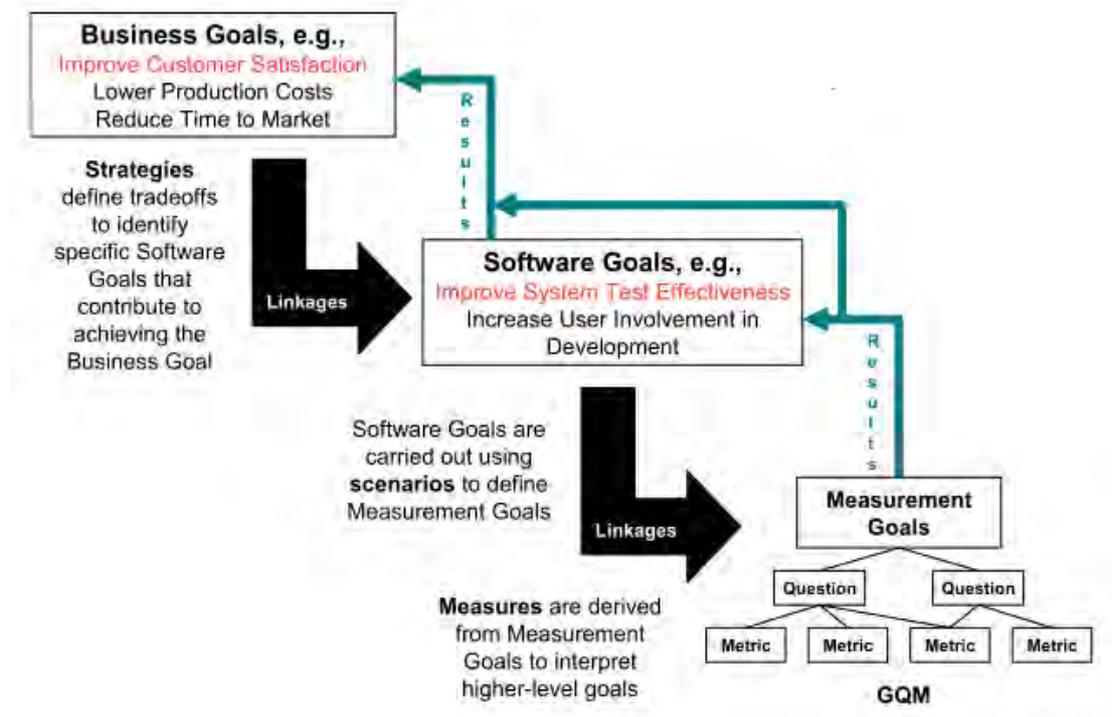


As KM is a combination of technology, culture and practices, management must put equal efforts into all the three aspects for successful implementation [6]. Along with the technology, management should promote a knowledge reuse/sharing culture within the teams and this must be practiced by making knowledge acquisition process continuous. Without a balanced approach for these three aspects, it will be hard to use KM to manage teams effectively.

Tips for better practices

Clearly identify business problems and align knowledge management project with overall business objectives.

Like any other software project, KM projects goals and objectives must be aligned to high-level objectives. This will give visibility to the management about how the KM activity is going to solve business problems. Let's see with an example how this is aligned by using the GQM (Goal Question Metrics) approach.



In the above diagram, business goals, strategies, and corresponding software goals were made explicit [8]. The business goal is “improve customer satisfaction”. This business goal might have been made by the organization due to the business problem such as so many post-release issues in the recent deliverable. To achieve this business goal and to solve the business problems, the software team sets its goal to “improve system test effectiveness”. This software goal might have been made by the management due to the problems within their test teams such as widening skills-gap, poor handling of changing customer requirements etc.

To achieve the software goal of “Improve system test effectiveness”, software project managers from their view point define their team’s goals, refine those goals down to specifications of data to be collected, and then analyze and interpret the resulting data with respect to the original goals. For this, a possible GQM goal could be

- **Purpose:** Improve
- **Focus:** the timelines of
- **Object:** software defect fixing
- **Viewpoint:** from the project manager’s view point
- **Context:** the characteristics of a software team in organization

This goal is then refined into specific questions that must be answered in order to evaluate the achievement of the goal. Relevant question and associated metrics related to the above goal are

- **Question:** What is the current software defect fixing speed?
- **Metrics:** Average life Cycle time of defect
 Standard deviation
 % cases outside the upper limit

At this level, the KM project measurements must be linked to the above metrics, so that KM project will be aligned with overall business objectives.

For example, the most tangible measurements of KM system involve who contribute or access which information. We need to track whether particular software development/test team members are regularly contributing and accessing the information. If yes, how many visits made to KM systems in a particular interval of time. If we can see the correlation between above metrics and KM system metrics, there will be significant impact of KM in project activities in achieving software goals. For instance, if there is negative correlation between number of visits and average cycle time of defect fixing, KM is making positive impact on the software teams.

Define clear roles and responsibilities to lead KM practices.

For successful implementation of KM, a well staffed team and strong leader, with clearly defined responsibilities are essential. Normally, the KM leader will be selected by the higher management, based on individual's expertise on project management and people skills. As a KM team leader, it is the responsibility of team leader to give guidance in organizing content and apply project management skills. In addition to this, he has the responsibility to improve his broad knowledge on various software sub-teams and start interacting with team members to contribute to KM. He also be attending various conferences and should bring latest trends of KM practices into the organization.

KM team members will be subject matter experts from each sub-team within the software division. They include members from software development team, testing team, automation team, etc. As a KM team member, he/she must be familiar with content and process involved. It's the responsibility of team members to categorize information efficiently. They must be monitoring content regularly and should assess the relevance of existing information. This is particularly useful if other teams looking for relevant information, they will be prevented from information overload.

There must be clear responsibilities defined to IS team members in KM projects. It's IS responsibility to analyze the existing systems and customize KM technical infrastructure. There will be cases of bottlenecks resulting from inadequate hardware or software, for which IS have the responsibility to resolve them.

The KM team roles outlined above are integrated in the project teams. There are so many advantages of having KM team like this. The biggest hurdle for many KM teams is identifying the team's tacit knowledge. If the KM team roles are integrated into project teams, identification of tacit knowledge within each team becomes very easy. KM team member of that particular project team will have good idea of individuals with tacit knowledge and the team members in need of particular knowledge. This helps the KM team to apply efficient ways of knowledge conversion from experience experts and the rest of the teams. And as the KM team member is integrated in the project team, he can assess the workload among individuals within his team and be able to find the right time to initiate knowledge conversion activities. For instance, when the project delivery date approaches, software test teams will be very busy with system testing activities. Forcing the test team members to be involved in KM activities at the peak times of execution might create negative impressions on the overall KM process. Software test team members will consider KM as additional work load imposed on them. The ideal time for these teams to share knowledge is during the break between each cycle of execution. Once the first cycle of execution completed, there will be little delay for the next build to arrive from development teams and to start the next cycle of execution. As a KM team member, from the same project team, he/she will be able to analyze efficiently what tacit knowledge from the earlier cycle of execution need to be shared and the suitable times for his/her team members to share the knowledge either into the KM system or with the rest of the team members directly.

There is a disadvantage of having KM team roles integrated into project teams. Normally, KM needs blend of people who have expertise in training skills, facilitation/influencing skills, communications skills and technology skills. In few cases, it becomes very difficult to find someone for KM team role with all these skills. Even when the KM roles are integrated in project teams, due to the lack of right people with right skills, KM will not have a major impact on the project activities.

To bring these additional skills to the team, like any other project management teams, KM teams can be filled with external consultants, who have expertise in dealing with KM related issues. There pros of engaging consultants in KM, such as

- Provide skills not within firm
- Counter internal constraint
- Mitigate risk
- Advise on IS/IT
- Overcome internal resistance to change
- Achieve change with greater speed than allowed by internal organic change

However, there are cons for this approach. In addition to higher costs, as the external people lack the idea of in-house procedures, there is possibility of developing gap between KM teams and project teams. The project team members understand KM team members as specialists who know nothing about their team's domain knowledge and required tacit knowledge need conversion. This will lead to the de-motivation of team members contributing to knowledge conversion activities.

If the required KM skill sets are available within the team or there is scope to develop these skills within the teams, it will be good to have KM roles integrated in project teams. Due to this, the gap can be narrowed b/w KM and rest of the teams. But, it will be hard to define what percentage of project activities should be used for formal KM, as it varies based on the size of team, amount of tacit knowledge need to be converted and the complexity of the required knowledge.

Implement KM in phased approach.

Similar to incremental approaches applied to various software development projects, KM also needs an incremental approach with different phases. This will reduce the amount of risk involved in these KM activities. You should divide the KM solution into various parts and address specific parts of KM solution in each phase. By laying foundation for next phase, each phase must provide immediate benefits and provide measurable ROI.

For instance, this can be made effective by addressing the need of unified access to existing information as part of the initial phase. And in the later phases, you address the need of improving the way knowledge is captured from various software sub-teams and managed.

Customize KM technical infrastructure to make it user-friendlier.

An organization KM system is the collection of information technologies used to facilitate the collection, organization, transfer and distribution of knowledge between various teams in software division.

Software organizations are using various technologies such as document libraries (ex: Google docs), Knowledge Bases (wiki, etc.), blogs, forums, SharePoint, etc.

A wiki is an extremely powerful KM tool for creating, maintaining and accessing knowledge bases. Since the introduction of the wiki technology in the early 2000s, many organizations have adopted the wiki for many of their knowledge bases. There are however a few disadvantages to wiki, as wiki platforms have a bit of a learning curve. Team members have to learn how to use it. If it becomes complex, team members will be reluctant to contribute to KM systems.

KM teams should know how to use wiki to their best effect and make the project team members aware of it. As everyone contributes to wiki, over a period of time it will end-up in mess, unless it's well organized. Information is often added to wikis but not deleted when no longer relevant or accurate or updated when changed.

This type of pros and cons exist for all technologies. For this reason, KM has to maintain and organize the technical infrastructure to make it user-friendlier. They can customize the KM system to rely on multiple technologies based on the team member's convenience. The success of KM depends on how well we customize technical infrastructure to make it easy to find the information from the KM systems. That means, if development team members are looking for some information related to test teams, they should easily find the information about the test team and more importantly the relevant information. Applications must be more focussed to the expected information and user friendly. If possible, it needs to blend in with the existing corporate systems such as organization intranet.

Encourage employees to contribute to KM by knowledge reuse and sharing.

As discussed earlier about Nonaka's model of knowledge conversion between tacit and explicit knowledge, software teams must be encouraged to contribute to these conversion activities.

Conduct as much knowledge sharing sessions as possible within the teams and across the teams. This can be conducted in various instances such as

- Whenever major issues noticed while development,
- New lessons learnt while carryout particular development/testing,
- Discussion on show stopper issues and the way to debug it

The management is responsible to decide which topic and how frequently the knowledge session should be held based on work priority. This is to ensure that the sharing sessions will really benefit the engineers and suits to their interest as well. The management must reward engineer's initiatives to contribute to such sessions.

The size of the organization or team will have the influence on knowledge sharing activities in terms of knowledge flow. As the size of an organizational unit increases, the effectiveness of internal knowledge flows dramatically diminishes and the degree of intra-organizational knowledge sharing decreases [9]. Management can overcome this by divisionalized organization structure or categorization of big teams into various sub-teams and using the appropriate user-friendly KM systems.

Knowledge re-use is another most important activity in achieving the benefits of KM projects. For example, if the software test teams are in need of any third party tools, management/team members must do basic assessment using KM systems before purchasing the tool. Assessment such as

- Is any other team working on similar feature in the organization?
- Are there similar tools used in different contexts?
- Are there any domain experts in this area? (use knowledge directories to find this)
- Can other team's knowledge be used in developing an in-house tool?

All these assessments are quickly possible with efficient KM system and routine knowledge reuse activities. In the above case, if team can reuse existing knowledge and develop in-house tool, it will save lot of money to the project and this tool can be re-used by other teams whenever required without any licensing issues like in third party tools.

As mentioned earlier, the size of the software organization influences the cost/benefits of the KM tools. For instance, small organizations or start-ups will not have sufficient funds to invest in KM tools. At the same time, if the size of the organization is very small, they don't have to rely much on KM tools and related technologies, but they can stimulate the knowledge flows using soft approaches such as face-to-face knowledge transfer, group sharing methods etc. Overall, organization management has to consider these aspects of cost/benefits with regards to size of the organizations before investing in KM tools or activities

Make knowledge acquisition process continuous.

KM is not an end itself. The knowledge acquisition process must be continuous in order to keep all knowledge up-to-date and ensure new knowledge is captured from time to time.

To support this, proper process needs to be defined as well, so that team members will record all related knowledge into the database. If a team member wants to share his tacit knowledge on a particular software, process, new methodologies, he should be given idea on the ways of sharing his knowledge with other team members, teams at organization level and ways of converting into explicit knowledge.

By doing this, knowledge acquisition task will be formalized within the software teams. This will help KM team to capture knowledge gained by the team members in a more effective way and continuously. While contributing to KM systems, another big challenge for the KM team is, whether to give free access to all project team members for contribution to KM systems? Or whether knowledge contributed to the KM tools will be "gated and selected" by the KM team?

In earlier sections, we have examined the benefits of organizations appointing KM roles integrated into project teams. In this context, it will be more beneficial, if every project team member is able to contribute their job related knowledge directly to KM systems. But, if the content of KM systems is not maintained properly, this will lead to information pollution and the users abandoning the systems.

To overcome this, KM team has to put more emphasis on content addition/modification/deletion at the initial stages of KM activities. And even deciding what kind of tacit knowledge must be entered into the systems. Once the project team members are trained on the format of contribution to KM systems and the relevance of appropriate tacit knowledge, KM team members can review the content regularly to verify the compatibility with the proposed KM

standards. At the initial stages, write permissions to KM systems are granted to few experienced project team members and based on the user activity and approval from KM team, it will be extended to rest of the team members.

Another aspect of managing the knowledge in KM systems, involves handling special cases, such as team members who leaves the company. KM team needs to be able to make sure other team members do not waste time trying to contact that person while preserving the knowledge they have contributed.

Review the KM project regularly

KM projects must be reviewed regularly to assess what is missing and finding ways to better organizing knowledge. The review begins by breaking the information into two categories:

1. What knowledge currently exists
2. What knowledge is missing

Once the location or source of the missing information is identified, KM teams can begin to structure the relevant information so that it can be easily found.

One of the common mistakes that most of the teams do while review process is to put more emphasis on cataloguing existing explicit information assets or the information that is documented, transferable and reproducible (ex: test reports, project proposals, ...). During the review process there should be however more emphasis on reviewing the cycles of knowledge conversion.

If the software test team has bought a particular third party component recently to test the features and debug the issues easily, in the review process they must review the efficient ways of how the tacit knowledge on the component translated to explicit knowledge. In addition to this, management must regularly review the alignment of current KM measurements with other project metrics and high-level goals.

Balance between technology, culture and practice.

For KM to be successful, as mentioned in earlier section, there must be a balancing act between technology, culture and practice. This can be achieved by placing the desire of people to use/involve in KM activities ahead of the technology. This needs cultural change within the teams and support from the management. For team members, if given the required time, training and incentives, they will begin to capture, manage and share knowledge with enthusiasm.

In addition to this, management must try to eliminate traditional rivalries between team members. This involves changing perception of employees: "to stay strong, I have got to hide and protect what I know." With this perception, most of the experienced team members will be reluctant to contribute to the KM activities. Here management has to play key role in changing the mindsets of the team members. Management has to assure that there will not be any threat to their positions by doing that. To resolve this, management has to create a supportive and collaborative culture by rewarding the individuals who contribute to KM activities. The performance appraisal criteria have to be changed to rate performance based on employee's cooperative efforts. Overall, cultural changes of this magnitude take time, so they have to practice this continuously to see the results of KM.

Share the success stories of KM practices at organization level.

Normally, two thirds of KM effort needs to focus on non-technical issues such as culture and practice. For making this effective, the management must use motivational approaches such as sharing success stories about KM implemented recently within their organization. If KM has been implemented successfully in software test development teams, the positive impact of KM within test development teams must be shared with other development teams, test teams and sustainable teams within the software divisions. These positive impacts could be how the tools development process has been improved, quick fixing of the tools issues, reduced attrition rates and shortening skills gap within the test development team. By doing this, it will stimulate the team member's curiosity to be involved in KM activities and help the KM team to apply KM methodologies in other parts of the organization.

Conclusion:

This article summarizes various challenges software organizations are facing in terms of global distributed teams, quick testing lifecycles, outsourcing models etc., and how KM can be implemented effectively with the best practices. To resolve the above mentioned issues software teams facing at the moment, KM practices make very high impact in long-term. For this, management/team members must be committed to implementing the above practices over a period of time to achieve better results. Like any other management methodology, there is no "one size fits all" type of method to implement Knowledge Management. As this whole process involves cultural changes of significant magnitude, teams must be patient and practicing this continuously to see the bottom line results.

References

1. Johnson, G and Scholes, K (2002) Exploring Corporate Strategy, 6th Edition, Prentice Hall
2. Ted E. Lee , "Applying Knowledge Management Approach For Software Testing"
3. Ikujiro Nonaka, "A Dynamic Theory of Organizational Knowledge Creation"
4. Ong Kein Wei, Tang Mei Ying, "Knowledge Management Approach in Mobile Software System Testing"
5. Anne Stuart, "5 Uneasy Pieces, Part 2, Knowledge Management," CIO Magazine, June 1, 1996
6. Tom Davenport, "Known Evils, Common Pitfalls of Knowledge Management," CIO Magazine, June 15, 1997
7. 2010 Global Most Admired Knowledge Enterprises (MAKE) Report -- www.knowledgebusiness.com/
8. Basili, Victor R; Mikael Lindvall, Myrna Regardie, Carolyn Seaman, Jens Heidrich, Jurgen Munch, Dieter Rombach, Adam Trendowicz (2010). "Linking Software Development and Business Strategy Through Measurement". Computer 43 (4): 57-65.
9. Alexander Serenko, Nick Bontis, Timothy Hardie, (2007) "Organizational size and knowledge flow: a proposed theoretical link", Journal of Intellectual Capital, Vol. 8 Iss: 4, pp.610 - 627

Erlang Open Telecommunications Platform

Francesco Cesarini, francesco [at] erlang-solutions.com
Erlang Solutions Ltd., <http://www.erlang-solutions.com/>

Telecommunications technology headlines are dominated these days by new mobile devices and their operating systems. The steady evolutionary march of radio access standards also grabs column inches as too do the latest innovations in operational/business support systems (OSS/BSS) solutions. Middleware developments seem to swing in under the media radar.

In fairness to the telco press, middleware developments can be bewildering and esoteric. Software development is perhaps not viewed as 'telco' so much as pure IT and so it is left to the very specialist press to cover innovations in the field. Middleware though is the unsung hero of telecommunications networks.

Choosing the right middleware can be the difference between an exciting new offering launching on budget and ahead of the rivals or damagingly late and ruinously expensive. With a wide variety of programming languages, such as Erlang, Java, C and C++ available, choosing the right middleware depends on a range of factors.

Erlang Programming Language

Erlang was originally invented by the Ericsson computer science lab as the programming language of choice for the next generation of telecom systems. While Erlang is a powerful programming language used to build distributed, fault tolerant systems with requirements of high availability, these complex systems require middleware in the form of reusable libraries, release, debugging and maintenance tools together with design principles and patterns used to style network architecture.

Ericsson realized this early and initiated a project to address issues in parallel with its first commercial project. Work began on the creation of the Open Telecom Platform, often referred to as OTP. Open, in this instance, stands for the openness of Erlang towards other programming languages, APIs and protocols. While Telecom was chosen when Erlang was only used internally within Ericsson for telecom products, years before it was released as open source. It might have made sense in the mid 90s, but today we say Telecom refers to the distributed, fault tolerant, massively concurrent soft real-time characteristics present in telecom systems, but valid in a wide range of other industry verticals. Platform refers to the use of OTP as middleware in complex systems.

Prior to its release as open source, OTP was used to develop many turnkey telecom solutions with millions of lines of code, including the AXD301 ATM switch, the GGSN IP gateway node and SGSN support node, two components handling core functionality in GPRS networks.

This approach resulted in a well-tested code base proven fit for the most demanding soft real-time systems.

But what exactly is OTP?

OTP can be seen as a control system platform for developing, deploying and running Erlang-based systems. Design principles provide software engineering guidelines that enable systems to be developed in a uniform way. Consequently, different programs that do completely different things in the network will have a common structures and functions.

When developing Erlang-based systems, it is not mandatory to use some or any parts of OTP when writing Erlang code. Using it, however, enhances productivity, reduces the overall code base and increases the code quality. It ensures developers do not reinvent the wheel for common design patterns and problems that have already been solved, hiding many of the tricky issues with concurrent and parallel programming.

OTP is made up of three components:

- The Erlang programming language
- A set of libraries, including tools, interfaces and reusable applications
- A set of design principles and patterns describing the system's architecture

OTP's design principles provide software engineering guidelines that enable developers to structure systems in a scalable and fault tolerant way. Different programs that execute completely different tasks will do so using common design patterns. While it is not mandatory to follow these design patterns, it is very advantageous to do so.

OTP building blocks

Java's key components are code threads which are mapped to the OS threads. While the key component when writing an Erlang program is known as a process, they are independent of OS threads. As a result, processes take microseconds to create and use little memory. They can be created and replicated with little overhead, enabling millions to interact concurrently on any system.

In Erlang, the most frequently used process patterns have been implemented in library modules, commonly referred to as OTP behaviours. They contain the generic code framework for concurrency and error handling, simplifying the complexity of concurrent programming and protecting the developer from many common pitfalls.

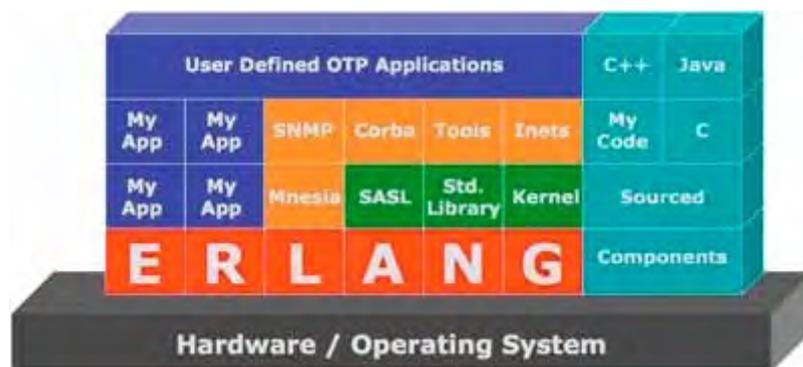


Figure 1 - Reusable architectures using OTP applications

Behaviours are monitored by supervisors, themselves a behaviour, and grouped together in supervision trees. A supervision tree is packaged in an application, creating a building block of an Erlang program.

Ready-made components are packaged as applications. They include databases, management protocol stacks, interfaces towards other languages, monitoring tools; components that can be reused in-between projects.

OTP applications that come with the standard Erlang distribution include the Systems Architecture Support Libraries (SASL), answering common maintenance and operations requirements including packaging, deploying and system upgrade during runtime.

Mnesia, a distributed, soft real-time database management system written in the Erlang programming language, enables transactions across hosts is another very popular application, alongside implementations of protocols and standards such as the simple network management protocol (SNMP), the Common Object Request Broker Architecture (CORBA) or the Interface Definition Language (IDL). For telecom systems, protocols such as Megaco H248 protocol stack are often complemented with proprietary components such as SIP or Diameter stacks.

A complete Erlang system such as the AXD301 switch or the SGSN and GGSN nodes is a set of loosely coupled applications that interact with each other. Some of these applications have been written by the developer, some are part of the standard Erlang/OTP distribution, and some may be other open source components. They are sequentially loaded and started by a boot script generated from a list of applications and versions. Take care not to confuse OTP applications with the more general concept of an application, which usually refers to a more complete system that solves a high-level task.

Erlang OTP's unique feature set provides an ideal platform for developing distributed systems.

Implementing similar systems in other languages requires a great deal of effort on features such as inter-node communication, message queues, failure detectors, and client-server abstractions.

OTP instead provides a battle-tested implementation of these facilities as part of its standard libraries, making it possible to rapidly prototype systems without significant overhead spent on the fundamental, generic building blocks.

OTP is more than a telecoms platform

In December 1998, a few years after the first internal release of OTP, the source code and libraries were released as open source, enabling Erlang to make headway outside of telecoms. New areas of use included banking, financial switches, web services, control systems or messaging frameworks such as instant Messaging and SMS, as they all have very similar characteristics to Telecom applications.

With embedded hardware becoming more powerful, Erlang/OTP is making headways in the embedded space, alongside cloud computing, noSQL databases and multiuser online gaming. These are all systems that have the same reliability and scalability requirements as telecom applications, and as such, benefitted from the features of OTP. OTP also made headways within telecoms outside of Ericsson and is used by companies such as T-Mobile, Nokia, Motorola and AT&T.

Some of the most successful OTP based projects today include EjabberD, an open source XMPP based instant messaging server with an estimated 40 per cent market share of the Jabber market, CouchDB and Riak, to popular databases in the NoSQL movement. Other users of OTP in the cloud computing environment include the Ruby On Rails Hosting companies Engine Yard and Heroku. Github uses Erlang in its infrastructure, as do Amazon and Clouant to scale their data stores. Nokia's disco project, a map-reduce framework in Erlang provides 90 per cent of Hadoop's functionality with 10 per cent of the code.

Less is more with OTP

There are many benefits to using OTP:

- Less code to develop as a result of the libraries and other components
- Developers have a common programming style and use a component based terminology
- OTP is a solid and well tested code base, so bugs are rare

All of this translates to time to market and lower cost of ownership through lower development and maintainability costs.

The productivity boosts when using OTP are visible in a study done by Ericsson where they estimated that using Erlang/OTP resulted in a four to ten-fold reduction in code compared to using Java or C++. The bug density and line of code productivity, however, remained the same, making them conclude that using Erlang together with OTP gave them a four to ten-fold increase in productivity and quality. These numbers were confirmed in another study at Heriot-Watt University, where rewriting C++ production code resulted in four to twenty fold code reduction . When examining what the individual lines of code, it was obvious that supervisors played a significant role. The defensive programming in Erlang consisted of one per cent of the total code base versus an average of 27 per cent for the C++ applications. Memory management consisted of 11 per cent of the total code base.

While the Erlang programming language is an excellent tool to build massively scalable distributed systems that will never go down, you need more than just a programming language to enable successful implementations. The Open Telecom Platform is the Erlang middleware that provides all of this. Providing interoperability between increasingly fragmented IT systems is about much more than supplying the glue that holds everything together.

Learn more about Erlang and the Open Telecom Platform on <http://www.erlang.org/>

Concordion

Tomo Popovic, tp0x45 [at] gmail.com

Concordion is an open source tool for writing automated acceptance tests in Java development environment. The main advantages of Concordion are based on its clean concept and simplicity. It is very easy to install, learn, and use.

Web site: <http://www.concordion.org>

Version Tested: Concordion 1.4.2

License & Pricing: Free, under Apache License, Version 2.0

Support: website and users group on yahoo: <http://tech.groups.yahoo.com/group/concordion/>

Introduction

Concordion is a very powerful tool for writing and managing automated acceptance tests in Java projects. Concordion directly integrates with JUnit, which allows for easy use with IDE of your choice (Netbeans, Eclipse, IntelliJ IDEA). One of the most appealing features is that Concordion uses acceptance tests specifications in native language, which allows use of Concordion for requirements management.

Installation

Installation of Concordion is very simple. You basically need to download and insert Concordion JAR file (comes with three dependency libraries provided). In NetBeans all the JARs go into the section for Test Libraries as shown in Figure 1.



Figure 1. Simple installation: inserting Concordion JAR files into the project

Alternatively, if you are using Maven, Concordion can be referenced in the POM file:

```
<dependency>
  <groupId>org.concordion</groupId>
  <artifactId>concordion</artifactId>
  <version>1.4.2</version>
</dependency>
```

Figure 2. Referencing Concordion using Maven

Using Concordion

Using Concordion assumes that developers understand and entertain the idea of active software specifications. Each feature or behavior needs to be specified, implemented, and verified by the means of active specifications and their connection to the system under development.

An active specification in Concordion consists of two key parts:

1. A nicely written requirement document describing desired functionality (XHTML). The XHTML specifications contain descriptions of the functionality illustrated with acceptance test examples. The examples data is marked using simple HTML tags.
2. Acceptance tests are written in Java and called fixture code. Tests are coded implementing a Concordion extension of a standard JUnit test case. Fixture code finds example data by marked by tags and use them to verify the system under development.

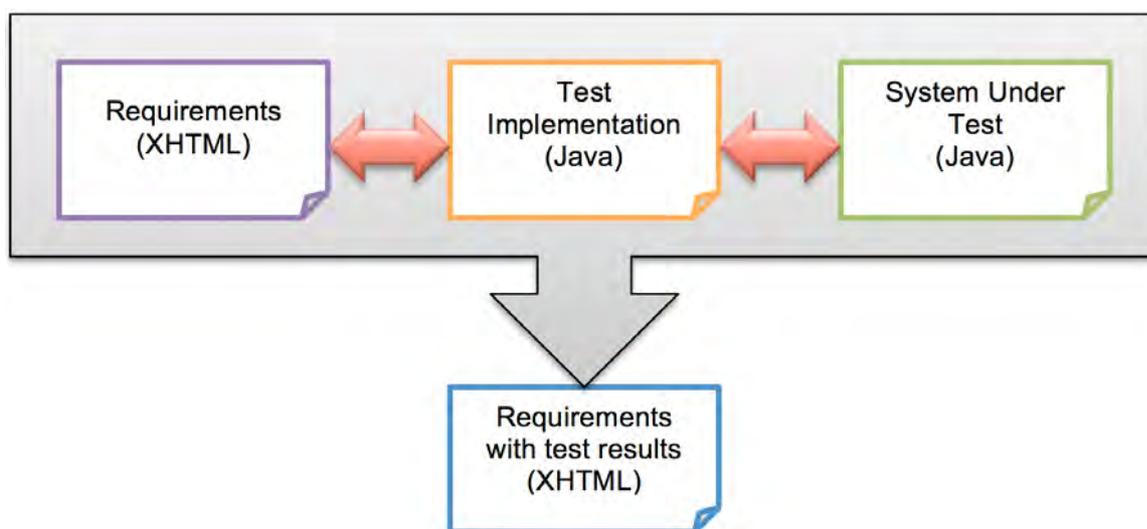


Figure 3. Concept of active specifications

The concept of using Concordion is illustrated in Figure 3. Acceptance tests are specified using native language and organized in Requirements (XHTML) files. Tests are implemented in Java and they connect examples from requirements with Java code of the system being tested. Test code is connected with the requirements through XHTML tags, which contain Concordion commands. This will be illustrated later in the article. Running tests in Concordion results in output XHTML files that combine the original specification and test results. Successful tests are highlighted “green” and unsuccessful with “red”.

This concept in Concordion is called active specifications due to a fact that test implementation links specifications and the system. Any change in the system will result in failed tests, which will remind us that we have to update the specification. Therefore, specifications never get old.

In order for Concordion to work, your project file structure needs to follow certain rules. In the example shown in Figure 4, we organize specifications, and tests implementation files into a package structure, here named **exampleapp.spec**.

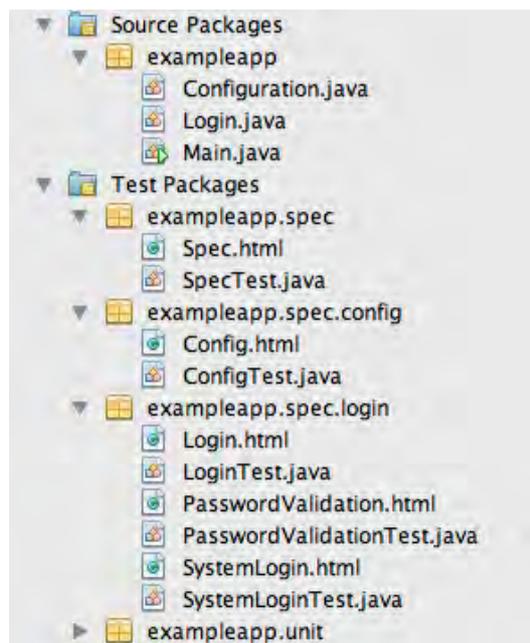


Figure 4. Organizing active specifications within Java project

As we can see in the example, other specifications and tests can be organized in sub-folders, which helps navigation. The folders structure also allows for easier navigation through the output files, which is implemented using “breadcrumbs”.

Simple example

To write specifications in Concordion, we use fairly simple XHTML syntax. In each specification document we need to use the “concordion” namespace at the top of the XHTML file. Please refer to the top of the example in Figure 5.

```
<html xmlns:concordion="http://www.concordion.org/2007/concordion">
  <head>
    <title>Config</title>
  </head>
  <body>
    <h1>REQ-003 Config</h1>
    <p>
      Provides feature for adding two integers. For example
      <span concordion:set="#a">2</span> plus
      <span concordion:set="#b">5</span> should give
      <span concordion:assertEquals="getTestResult(#a,#b)">7</span>.
    </p>
  </body>
</html>
```

Figure 5. A simple example of active specifications: Config.html file

Further in the example, we use simple XHTML tags to mark specific parts of the example and connect them to our test implementation code, which is in Concordion also called fixture code. Variables **#a** and **#b** are set to 2 and 5 inside of the sentence “For example 2 plus 5 should give 7” using **concordion:set** command. Another tag is specifying **concordion:assertEquals** command, which as a parameter in this case invokes internal method implemented in the corresponding test class. For more information on Concordion tags and commands please refer to Concordion website, which provides an excellent tutorial. Java fixture code for this active

specification is given in Figure 6.

```
package exampleapp.spec.config;

import exampleapp.Configuration; // connects to the System Being Tested
import org.concordion.integration.junit3.ConcordionTestCase;

public class ConfigTest extends ConcordionTestCase {

    public int getTestResult(int a, int b) {
        Configuration conf = new Configuration();
        return conf.addTwoIntegers(a,b);
    }
}
```

Figure 6. Test implementation (fixture code): ConfigTest.java

The implementation class uses name same as the XHTML specification, but with suffix Test, in this case **Config.html** and **ConfigTest.java**. The test connects the actual system being tested with the XHTML active specification. An instance of Configuration class is created, and then its method that adds two numbers is called. The result is passed back to the active specification. Upon executing the tests, the Concordion generates an output XHTML file, with inserted red or green highlights around the test results.



Figure 7. Test results: number 7 highlighted “green”

Providing tests in tables

Sometimes we need to run several test cases in order verify the desired behavior. In automated acceptance test tools this is typically done via tables. Concordion is not an exception. We can specify a test data set using standard XHTML tables and then use Concordion command set applied to table headers, in form of tags, to mark the table data for use in testing. Concordion will grab the test data from each row in the table, run the acceptance test and compare the results against the expected output. This feature is very useful when there is a need to run several testing data sets.

```

<html xmlns:concordion="http://www.concordion.org/2007/concordion">
  <head>
    <title>System Login</title>
  </head>
  <body>
    <h1>REQ-001 System Login</h1>
    <p>
      The system shall provide system logon function that will be used when
      a user attempts to use the system. The user needs to provide credentials
      in a form of username and password pair.
    </p>
    <div class="example">
      <h3>Examples</h3>
      <p>
        For the purpose of demonstration the system under test contains a
        system login function with username and password hardcoded to
        "johndoe" and "123abc!@#" respectively. All other combinations
        should fail.
      </p>
      <table concordion:execute="#valid=systemLogin(#username, #password)">
        <tr>
          <th concordion:set="#username">Username</th>
          <th concordion:set="#password">Password</th>
          <th concordion:assertEquals="#valid">Sucess</th>
        </tr>
        <tr>
          <td>john</td>
          <td>doe123abc!@#</td>
          <td>no</td>
        </tr>
        <tr>
          <td>admin</td>
          <td>admin</td>
          <td>no</td>
        </tr>
        <tr>
          <td>johndoe</td>
          <td>123abc!@#</td>
          <td>yes</td>
        </tr>
      </table>
    </div>
    <h2>Further Details</h2>
    <ul>
      <li>How to create user and password?</li>
      <li>Username restrictions and validation?</li>
      <li><a href="PasswordValidation.html">
        Password restrictions and validation?</a></li>
      <li>Can email be used instead of username?</li>
    </ul>
  </body>
</html>

```

Figure 8. An example of test data in a form of table – PasswordValidation.html

```

package exampleapp.spec.login;

import org.concordion.integration.junit3.ConcordionTestCase;
import exampleapp.Login; // system under design

public class PasswordValidationTest extends ConcordionTestCase {

    public boolean isValid(String password) {
        Login login = new Login();
        return login.validatePassword(password);
    }
}

```

Figure 9. Test implementation (fixture code) - PasswordValidationTest.java

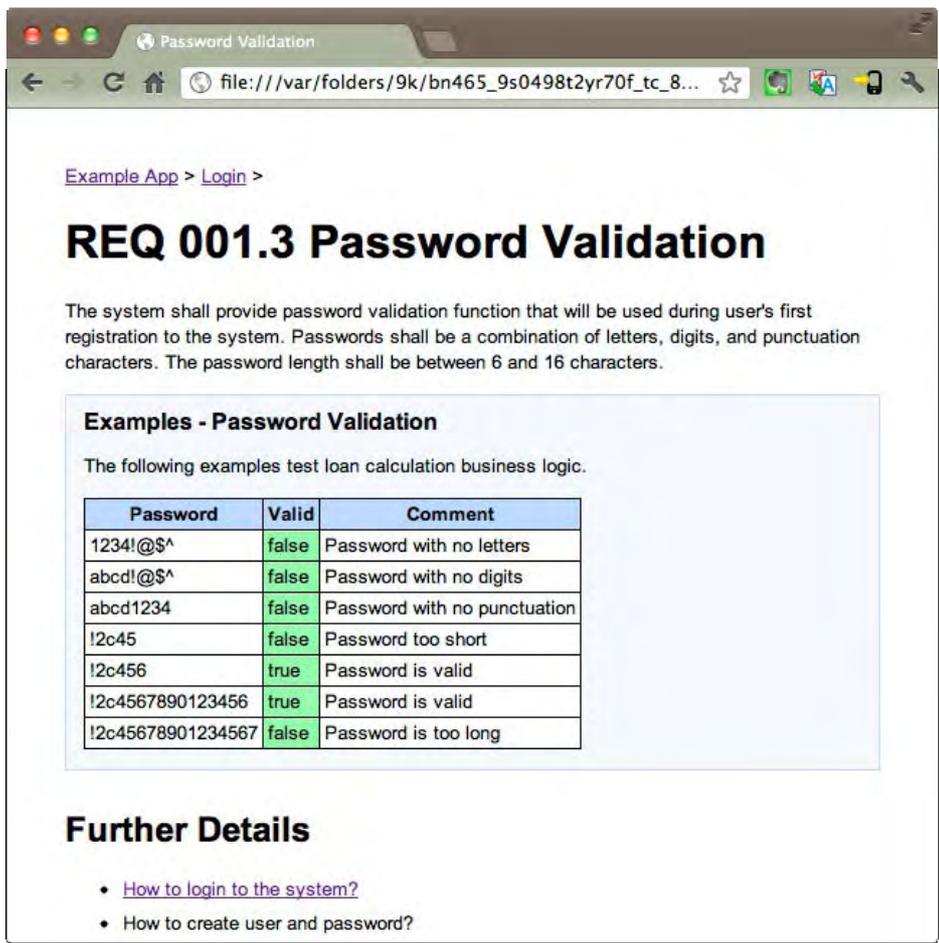


Figure 10. Tabular test output: successful runs highlighted “green”

An additional benefit of using XHTML files for specifications is that we can use hyperlinks between the specifications. In the example above, this is illustrated with “Further Details” section that links to Password restrictions and validation.

Suite of Tests

It is very wise to organize and structure our specifications and tests nicely. In Concordion, this comes very naturally for Java developers to organize specifications and tests into packages. In addition, Concordion offers a very nice way of grouping related tests into groups or subgroups and, therefore, create test suites. As shown in the example in Figure 11, a test suite can be created by simply referencing other XHTML test specifications.

```
<html xmlns:concordion="http://www.concordion.org/2007/concordion">
  <head>
<title>Example App</title>
  </head>
  <body>
    <h1>Example App</h1>
    <p>
      <a concordion:run ="concordion" href="login/Login.html">
        REQ-001 System Login</a>
    </p>
    <p>
      REQ-002 Browse System Events
    </p>
    <p>
      <a concordion:run ="concordion" href="config/Config.html">
        REQ-003 Update System Settings</a>
    </p>
  </body>
</html>
```

Figure 11. Suite of tests generated by referencing XHTML files (Spec.html)

```
package exampleapp.spec;

import org.concordion.integration.junit4.ConcordionRunner;
import org.junit.runner.RunWith;

@RunWith(ConcordionRunner.class)
public class SpecTest {

}
```

Figure 12. Empty class that implies running of tests in suite (SpecTest.java)

An empty class shown in Figure 12 implies running of tests in the suite. Please note that tests in the suite need to be organized in proper sub-packages as shown previously in Figure 4. This organization of files also produces nice “breadcrumb” navigation links at the top of the output pages (please refer to the top of the page in Figure 10 above). Hyperlinks in the result file will be highlighted red or green depending on the outcome of the test run (Figure 13).

Extensions

Concordion extensions allow for adding new functionalities. Extensions enable users to add their own commands, listen to events, and modify the Concordion output. The extensions are installed from separate JAR. The installation and use of extensions is beyond the scope of this article, but it is important to mention them. Please refer to Concordion website for more information.

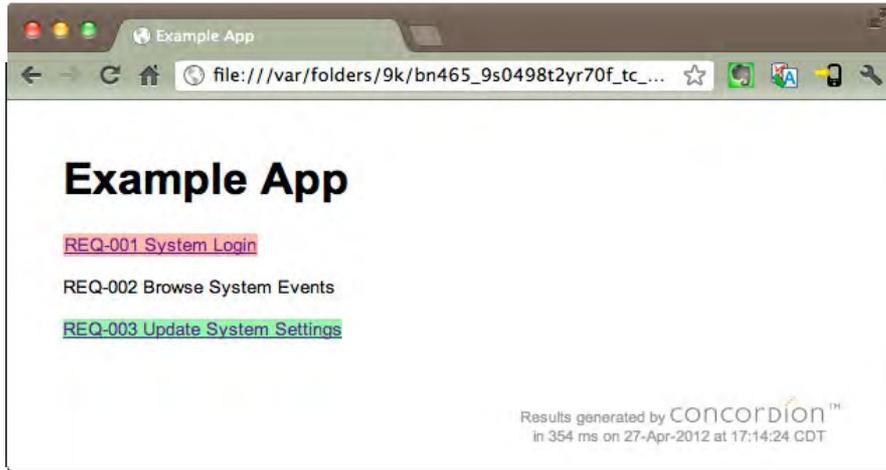


Figure 13. Test suite: highlighted hyperlinks indicate success/fail of tests

Conclusions

Concordion is a very interesting open source tool for automating acceptance testing in your Java projects. It is very easy to install and it has fairly fast learning curve. It can easily be used with various IDEs. If used properly and pragmatically, it can be a very good tool for managing the requirements specifications as well. Since the specifications are linked with the system, any change in the system or specifications will result in failed tests, which will prompt us to keep the specifications up-to-date and in sync with the system. Therefore, our specifications never go stale.

Concordion was originally developed for Java, but now there are also versions for .NET, Python, Scala, and Ruby. Please check the Concordion website for latest development.

Further reading

- Concordion website: <http://www.concordion.org>
- Users group: <http://tech.groups.yahoo.com/group/concordion/>
- Lisa Crispin, Janet Gregory, “Agile Testing: A Practical Guide for Testers and Agile Teams”, Addison-Wesley, 2009

Mockito

Tomek Kaczanowski, kaczanowski.tomek [at] gmail.com
<http://practicalunittesting.com>

Mockito is a popular open source Java mocking framework. It is very powerful and easy to use. It gives a lot of power into the hands of developers, but at the same time does not corrupt their hearts! Instead it urges them to write clean, well-designed and loosely-coupled code.

Web Site: <http://mockito.org>

Version Tested: Mockito 1.9.0 on Linux with Java 1.6.0_26

License & Pricing: Open Source (MIT License, <http://www.opensource.org/licenses/mit-license.php>)

Support: User mailing list <http://groups.google.com/group/mockito>

Using Mockito

Mockito is distributed as a single JAR file, which must be available in the classpath in order to execute tests. Usually this is provided by a properly configured build tool like Ant or Maven.

Why Test Doubles?

When writing tests it is often required to control the environment in which tested object (a so-called system under test (SUT), which can be a single class, a module or an application) is running. It is especially important to isolate the tested class in unit tests to make sure that it works as expected in every circumstances. It is often also required to verify if the tested object interacts with its collaborators in a particular way. Mock objects (or to be more precise - test doubles) allow to achieve both goals.

Consider a simple class - named `WeatherForecast` - which fetches weather forecasts from some external sources (e.g. web services). Now imagine writing tests for this class. It would be quite hard to test all its logic if we had to use the real collaborators (real web services). How can we make a real web service that will provide all the needed answers to test all possible paths of execution and thus test every bit of the `WeatherForecast` implementation? How can we verify how `WeatherForecast` class behaves when its collaborators behave unexpectedly - e.g. by breaking connections, responding with nonsense data or throwing exceptions? Making real web services behave in such awkward way might be really hard, if it is possible at all! Mock objects (or to be more precise stubs) allow us to test such classes, which rely on “external” collaborators.

The usefulness of mock objects is not limited to handling of such “external” collaborators. Following the Test-Driven Development (TDD) approach, we often write classes which use collaborators (in form of interfaces) that are not yet implemented. This is also a perfect place for using test doubles (mocks and/or stubs) to test this emerging design.

NOTE: Please note, that even testing with test doubles is usually not enough to prove your application is working correctly! Make sure you have also some end-to-end tests which test “the real thing”!

Mockito's Features

This short introductory article will describe what values Mockito can bring to your testing activities. I will concern less with the syntax which is described in details on Mockito's homepage and numerous websites.

Apart from "typical" tasks of a mocking framework (i.e. creation of test doubles, stubbing, setting expectations and verification of behavior), Mockito has some features which distinguish it from other mocking frameworks:

- Mockito allows to write test methods compatible with "arrange/act/assert" approach. This is different from what other mocking frameworks require, and feels much more "natural".
- Mockito can be used to write Behavior Driven Development (BDD)-style tests with some syntactic sugar that facilitates it.
- Mockito provides a nice, easily readable syntax. It also provides some annotations useful for reducing boilerplate code.
- It is easy to read Mockito's error messages. They also point out to possible mistakes a developer can make, which is very handy for people who begin to work with Mockito.

A lot of attention is given to the maintainability of tests. This is easier to achieve with Mockito than with other frameworks, by its following properties:

- Mockito is "forgiving" by defaults as it verifies only the interactions that it was asked to verify. This allows to write very focused tests that are not fragile. Mockito also makes stubs return some canned values by default like zeros, empty strings and falseys.
- Mockito allows writing of relaxed tests. Developers can specify what is really important for the given test scenario, and abstract over the unimportant details, using constructs like `anyString()` instead of specific values for instance.

It is worth mentioning that Mockito allows also for verification of quite sophisticated scenarios. For example, it can be used to:

- verify the number of method calls (using its `times()` method),
- verify the order of calls to particular collaborators (e.g. first, method `a()` was called on collaborator `A`, then method `b()` was called on collaborator `B`),
- inspect the arguments of methods created within the tested methods (using `ArgumentCaptor` class),
- apply "partial mocking" technique.

Some of the above should be use with extreme caution as they can hurt tests readability and/or maintainability! Mockito's documentation explains this in detail.

An Example

As an example of Mockito let us consider the following, simple class:

```
public class WeatherForecast {
    private WeatherService globalWeather; [1]
    private WeatherService localService; [1]

    public WeatherForecast(WeatherService globalWeather, WeatherService
localService) {
```

```
        this.localService = localService;
        this.globalWeather = globalWeather;
    }

    public Weather getForecast(String city) { [2]
        if (localService.hasForecastFor(city)) {
            return localService.getWeather(city);
        }
        return globalWeather.getWeather(city);
    }
}
```

This simple class has two collaborators that are both of the `WeatherService` type [1]. When asked for a weather forecast [2], it first queries the local forecast service, and only if it can not provide a forecast, refers to the global weather service. This logic is quite typical for the citizens of object-oriented systems, in which each class has a very limited responsibility and often relies on others to perform its duties.

Now let us see a test code, which verifies one of the possible execution paths of the `WeatherForecast` class.

```
@Test
public void
shouldFetchWeatherForecastFromGlobalServiceIfNotAvailableLocally() {
    // creation of collaborators [1]
    WeatherService localWeatherService =
Mockito.mock(WeatherService.class);
    WeatherService globalWeatherService =
Mockito.mock(WeatherService.class);

    // creation of SUT [2]
    // and injection of collaborators
    WeatherForecast forecast = new WeatherForecast(globalWeatherService,
localWeatherService);

    // stubbing of collaborators
    // telling them what should they do when asked [3]
    Mockito.when(localWeatherService.hasForecastFor(anyString()))
        .thenReturn(false);
    Weather forecastedWeather = new Weather();
    Mockito.when(globalWeatherService.getWeather(anyString()))
        .thenReturn(forecastedWeather);

    // invocation of the SUT method [4]
    Weather weather = forecast.getForecast("myCity");

    assertThat(weather).isNotNull(); [5]
    assertThat(weather).isSameAs(forecastedWeather);
}
```

This test presents all the typical phases of Mockito test. First the collaborators are created [1]. Mockito's `mock()` method is used (in reality I would rather import it statically so the `Mockito.` part could be omitted). It results with the creation of an imposter who looks like a real object of the `WeatherService` type, but can be strictly controlled. This is exactly why we use Mockito to create it.

[2] Then the object we attempt to test - of the `WeatherForecast` type - is created. Its collaborators are injected via its constructor.

Now, to test a specific scenario we need to instruct the collaborators on what they should do. Let us test that if the local weather service is unable to provide a forecast, then a global service is used. In order to do this, we need to inform both weather services about their expected behavior [3].

After every object knows its role, we can call the actual method of the SUT [4] and verify that it behaves properly [5].

Now comes the verification part. In this case, it is enough to verify that the returned `Weather` object is the same as the one which the global service should provide.

As we can see the expectations part [3] are really readable. For example one can read the following line:

```
Mockito.when(localWeatherService.hasForecastFor(anyString())).thenReturn(false);
```

like this:

”When someone asks you, `localWeatherService` if you can provide weather forecast for any city, you will answer with no.”

Please note that this test is very careful not to verify too much. The main reason for this is not to bind the test too strongly to the current implementation of the `WeatherForecast` class. If required, we could use one of the methods provided by Mockito to strictly control what interactions happened during the execution of SUT methods. We could for example write the following additional assertions:

- `Mockito.verify(globalWeatherService).getWeather("myCity");` - to verify that the `getWeather()` method was actually called on `globalWeatherService` collaborator.
- `Mockito.verifyNoMoreInteractions(localWeatherService)` - to make sure that apart from calling the `hasForecastFor()` of the `localWeatherService` collaborator, its no other methods were called.

Such verifications are often required but should be used with caution. As a rule of thumb, whenever possible we should use state testing and not interactions testing that verifies the methods called on collaborators. This means that we should assert on the returned values or changed states of objects.

Limitations

Even if Mockito is very powerful, useful and easy-to-use, it still has some limitations. They result from the conscious decision of its designers who created a tool aimed at working with well-designed code. Mockito shines when used against well encapsulated, loosely-coupled code. However, if you work with a dreadful spaghetti code filled with tight-coupled classes, Mockito will not provide means sufficient to deal with it. There is no support for mocking of static or final classes. Some will say this is a weakness of Mockito, while others will claim that this is an important feature of this tool, which makes developer write better code or redesign/refactor existing one.

Conclusion

If you look at the frameworks' popularity, you will notice that Mockito is very popular in Java world (if not a number one!), and still gaining popularity. It has already proved its usefulness and robustness in thousands of open-source and commercial projects. It is frequently released, and is still being improved. It can be used with other popular tools like JUnit, TestNG, Cobertura, Hamcrest or FEST Fluent Assertions. Last, but not least, it has a very friendly community with an active mailing list, and impressive documentation.

I would like to end this article with a quote from Mockito's website, which very well describes this superb tool: *“Mockito is a mocking framework that tastes really good. It lets you write beautiful tests with clean & simple API. Mockito doesn't give you hangover because the tests are very readable and they produce clean verification errors.”*

Further Readings

- Mockito documentation <http://docs.mockito.googlecode.com/hg/latest/org/mockito/Mockito.html>
- Mockito mailing list <http://groups.google.com/group/mockito>
- Practical Unit Testing with TestNG and Mockito, book by Tomek Kaczanowski, <http://practicalunittesting.com>
- Szczepan Faber's blog <http://monkeyisland.pl/>

Robotium

Renas Reda

Robotium is a test framework created to make it easy to write powerful and robust automatic UI test cases for Android applications. With the support of Robotium, test case developers can write function, system and acceptance test scenarios, spanning multiple Android activities. Robotium tests can be run on both emulator and device.

Web Site: <http://Robotium.org>

Version tested: 3.2.1

System requirements: Android SDK

License & Pricing: Free, Apache 2

Support: Issue tracker at <http://code.google.com/p/robotium/issues/list>,

Mailing list at <https://groups.google.com/forum/?fromgroups#!forum/robotium-developers>

Benefits of Robotium

- Easy to write
- Shorter code
- Automatic timing and delays
- Automatically follows current Activity
- Automatically finds Views
- Automatically makes own decisions, e.g. when to scrolls
- Test execution is fast

Installation

Robotium is downloaded as a single JAR file, which can be placed in any folder. To actually start using Robotium, the JAR must only be put in the classpath of the test project.

Robotium can be used with all the different build automation tools. Perform the below steps to use Robotium with Maven:

1. Set up your project(s) as usual for Maven, see [maven-android-plugin](#).

2. Add a dependency to Robotium in your **test** project:

```
<dependencies>
<dependency>
<groupId>com.jayway.android.robotium</groupId>
<artifactId>robotium-solo</artifactId>
<version>3.2.1</version>
</dependency>
...
</dependencies>
```

NB: For the version number, enter the version number of the latest release!

Usage

Robotium can be used both for testing applications where the source code is available and applications where only the apk is available (implementation details not known). For more information, instructions, and step by step tutorials see the Robotium website.

To use Robotium, create an Android tests project and add a test class. Usually the `ActivityInstrumentationTestCase2` test class is used to write Robotium test cases. However, Robotium is compatible with all the Android test classes. When writing Robotium test cases only one class is used: Solo.

The below example illustrates how a Robotium test case might look like when used in conjunction with `ActivityInstrumentationTestCase2`:

```
public class EditorTest extends ActivityInstrumentationTestCase2<EditorActivity> {
    private Solo solo;
    public EditorTest() {
        super("com.test.editor", EditorActivity.class);
    }
    public void setUp() throws Exception {
        solo = new Solo(getInstrumentation(), getActivity());
    }

    public void testPreferenceIsSaved() throws Exception {

        solo.sendKey(Solo.MENU);
        solo.clickOnText("More");
        solo.clickOnText("Preferences");
        solo.clickOnText("Edit File Extensions");
        Assert.assertTrue(solo.searchText("rtf"));

        solo.clickOnText("txt");
        solo.clearEditText(2);
        solo.enterText(2, "robotium");
        solo.clickOnButton("Save");
        solo.goBack();
        solo.clickOnText("Edit File Extensions");
        Assert.assertTrue(solo.searchText("application/robotium"));
    }

    @Override
    public void tearDown() throws Exception {
        solo.finishOpenedActivities();
    }
}
```

Some Robotium commands

- `getCurrentActivity()`
- `clickOnButton(String regex)`
- `clickInList(int line)`
- `enterText(int index, String text)`
- `searchText(String regex)`
- `waitForText()`, `waitForActivity()`, `waitForView()`
- `clickOnMenuItem(String text)`
- `goBack()`, `goBackToActivity(String name)`

More information

More information on how to use Robotium can be found on the website. An example test project as well as various step by step tutorials with images can be downloaded from [Robotium.org](http://robotium.org).

Gettings started: http://code.google.com/p/robotium/wiki/Getting_Started

Tutorials: <http://code.google.com/p/robotium/wiki/RobotiumTutorials>

JMeter-Plugins - More Obvious and Powerful Load Testing

Andrey Pohilko, JP@GC, <http://code.google.com/p/jmeter-plugins/>

JMeter Plugins at Google Code (JP@GC) is a popular third-party plugins set for JMeter, extending its functionality with a dozen of graphs, new load delivery controllers and other functions that are missing from the original JMeter package. Basically, the JP@GC set contains two types of plugins: graph plugins and JMeter functionality extensions.

Web site: <http://code.google.com/p/jmeter-plugins/>

Version Tested: JMeter-Plugins 0.5.2 as of May 14, 2012

License & Pricing: Free & Open Source

Support: Users Mailing List <http://groups.google.com/forum/#!forum/jmeter-plugins>

Documentation: Online <http://code.google.com/p/jmeter-plugins/>

Installation

To add extensions to JMeter, you should simply locate the *lib/ext* folder under JMeter installation path and extract the JP@GC distribution package contents into it. The latest JP@GC package can be always found at the project page <http://code.google.com/p/jmeter-plugins/>

Restart JMeter to load the installed plugins and look for items named like «jp@gc - ...» in JMeter menus. Use those items as if they are regular JMeter elements.

Documentation

Every JP@GC plugin has an immediate «Help on this plugin» link, opening Wiki page containing description for current extension in Web browser. If you still confused or have some tricky issue — feel free to ask at project support mailing list.

Graph Plugins

The JMeter tool itself has a sound position in the world of performance testing, because it offers professional level features, comparable to commercial tools, and still stays free and open source. However, out-of-the-box JMeter has some weak points, mainly a lack of reporting features and visualizations.

JP@GC targets this weak point and offers several groups of test results visualization plugins. All of these plugins can be found in *Edit => Add => Listener* menu of JMeter. Every graph plugin in JP@GC has three tabs: *Chart*, *Rows* and *Settings*. Don't forget to play with controls at *Settings* tab — there are a lot of look-and-feel settings to get the best view of your data. Also consider right-clicking the graph area, there is popup menu for saving image or CSV data under it.

Target Resource Monitoring

PerfMon Metrics Collector is the most popular component of the jmeter-plugins set. It allows to monitor the resources usage of the target computer within JMeter. It requires a special ServerAgent process to be started at target computer and plots collected data on nice graph (Figure 1).

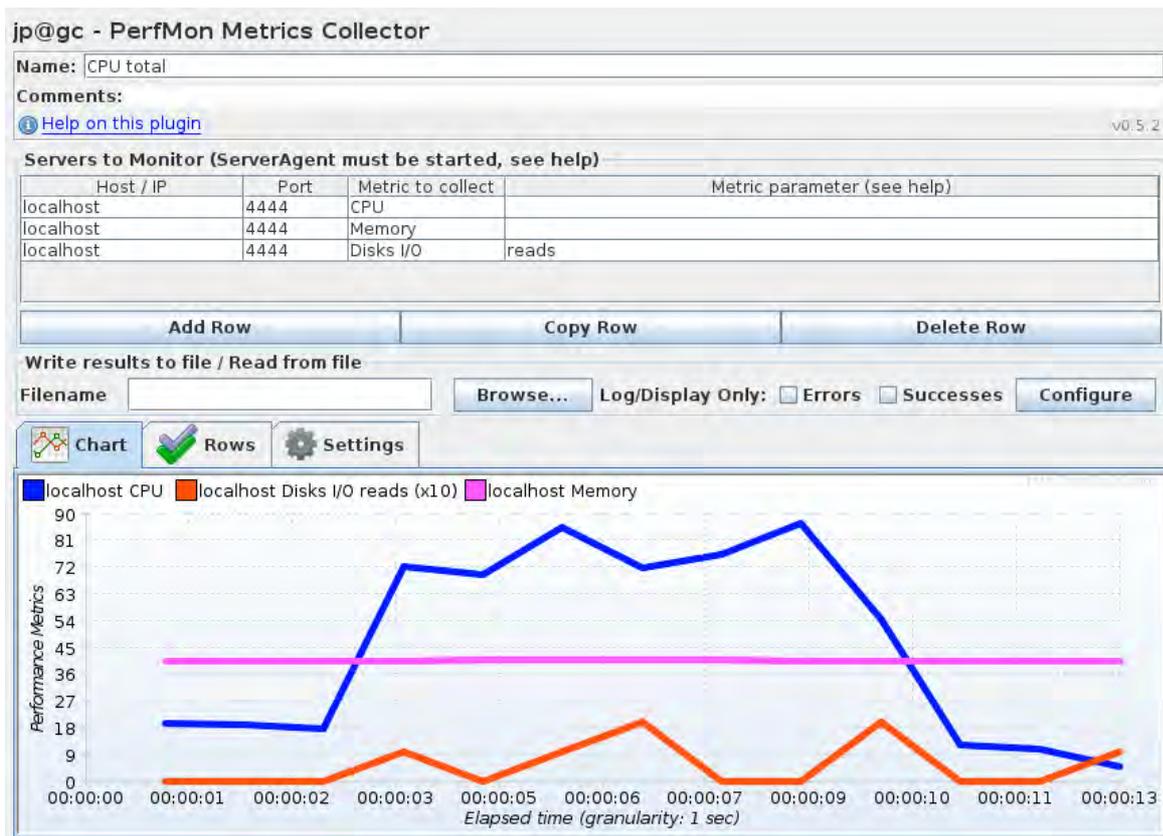


Figure 1 - Target Resource Monitoring Plugin

PerfMon plugin supports collecting an unlimited number of metrics at the same time. It can collect over 75 separate metric types in groups: CPU, Memory, Disk, Network, Swap, TCP statistics, Java JMX metrics. Per-process and system total metrics are supported. In case you need to collect a custom metric that is calculated by your program/script, you may use the *Custom* metric type.

Timelines

The most numerous graphs in JP@GC are timelines which have all names like «... Over Time» or «... per Second». Timeline graphs visualize the results during the test run, and also can load non-GUI JMeter test results from JTL files. The available timeline data types to plot are: response times, server hits (transactions) per second, response codes, active JMeter thread counts.

Composite Graph is a special plugin that does not collect data itself, but uses other timelines as a source. You can aggregate different data in one graph to track correlation between test parameters and examine how response time depends on active threads count for instance. Since *PerfMon Metrics Collector* has also a timeline type, it can be used as a source for *Composite Graph*, too.

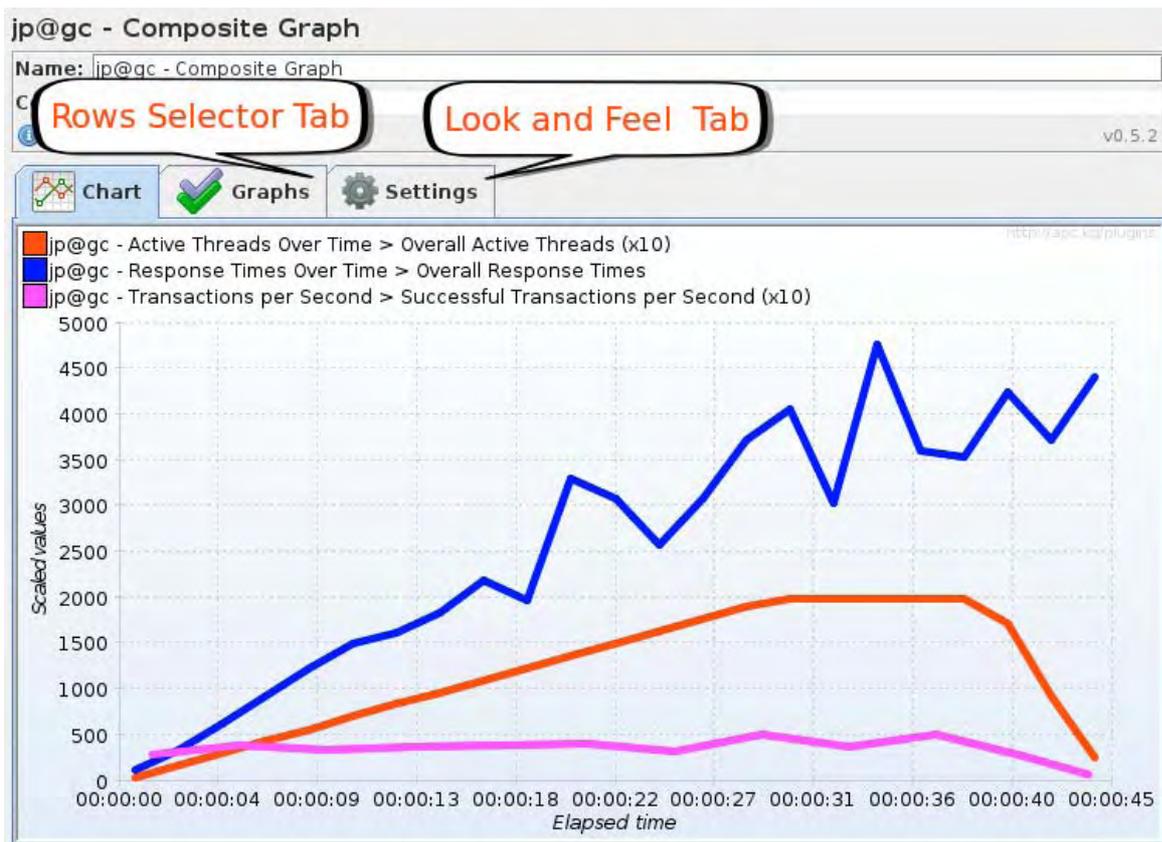


Figure 2 - Composite Graph with three different graphs chosen

Distributions

Distribution analysis is vital in performance testing, so JP@GC has a pair of graph plugins for distribution analysis: *Response Times Distribution* and *Response Times Percentiles*.

Distribution bar chart is a commonly used analysis technique and JP@GC provides flexible graph with configurable granulation (Figure 3). There is special «Aggregate Mode» switch at settings tab. If the aggregate mode is selected, all data rows are merged into single row, showing whole test response times distribution.

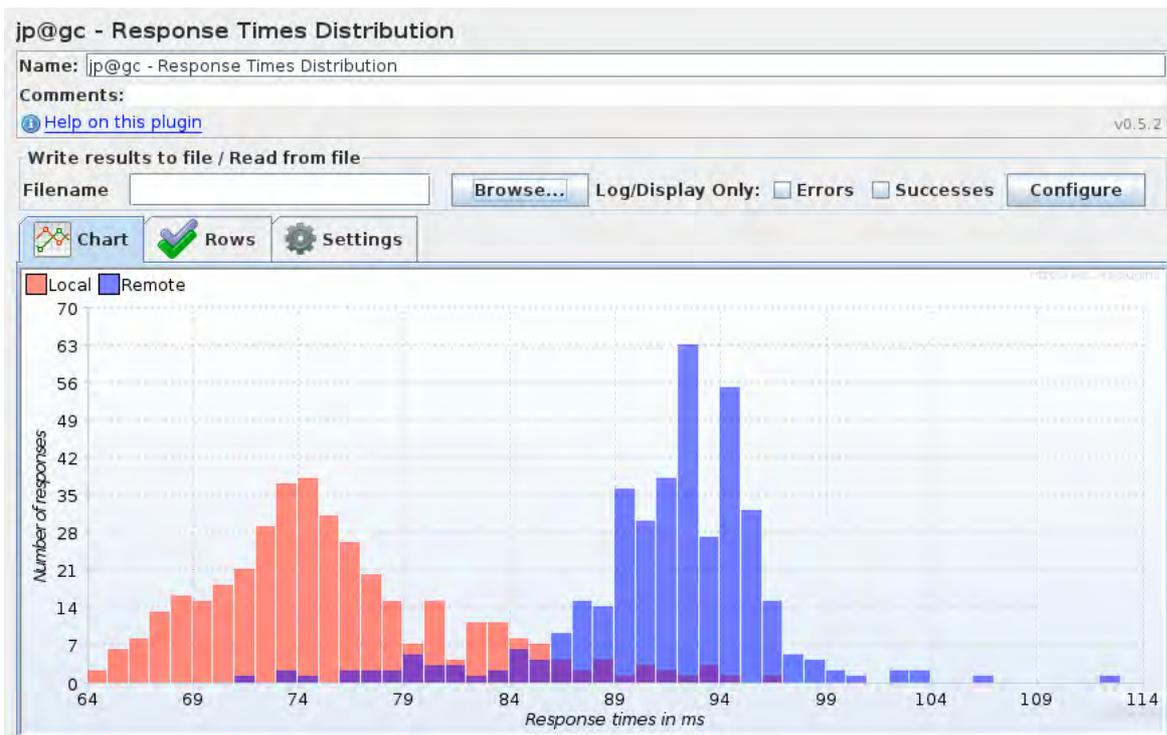


Figure 3 - Response Times Distribution visualized

The *Percentiles* graph is used for advanced results analysis., It allows analyzing not only 50% or 90% line, but all other percentile levels that can be set in external SLA requirements.

Command-Line Tool

Advanced JMeter users rarely perform regular JMeter tests via GUI, they prefer to set up tests and run them in non-GUI mode, via command-line. To help users automate their test reporting, JP@GC has a command-line tool that can consume JTL files and produce the same graphs as JMeter GUI plugins. Also it can be used to export graph data to CSV file for further processing. For example, the following command:

```
JMeterPluginsCMD.bat --generate-png test.png --input-jtl results.jtl --
plugin-type ResponseTimesOverTime
```

will generate the same graph as if *results.jtl* file would be loaded in *Response Times Over Time* plugin from JMeter GUI.

Feature Extension Plugins

Visualization is not the only area where JMeter functionality can be efficiently extended. There are also useful additions for load supply, logging, advanced inter-thread parameter passing and many others. Only the key feature extensions will be described in this article, all others are available in project's documentation.

Load Delivery Plugins

The original JMeter package contains only one load supply controller: simple Thread Group. However, many users need solution to configure threads to start by bunches, or to create complex load supply scenario. JP@GC provides two solutions: *Stepping Thread Group* and *Ultimate Thread Group*. For those users who use not thread-driven, but request-per-second-driven load testing, there *Throughput Shaping Timer* plugin.

Stepping Thread Group (STG) allows to set up users activation scenario where threads start with «stepping» schedule. Aiming to be more obvious than original JMeter elements, the preview graph in STG shows expected active threads count (Figure 4).

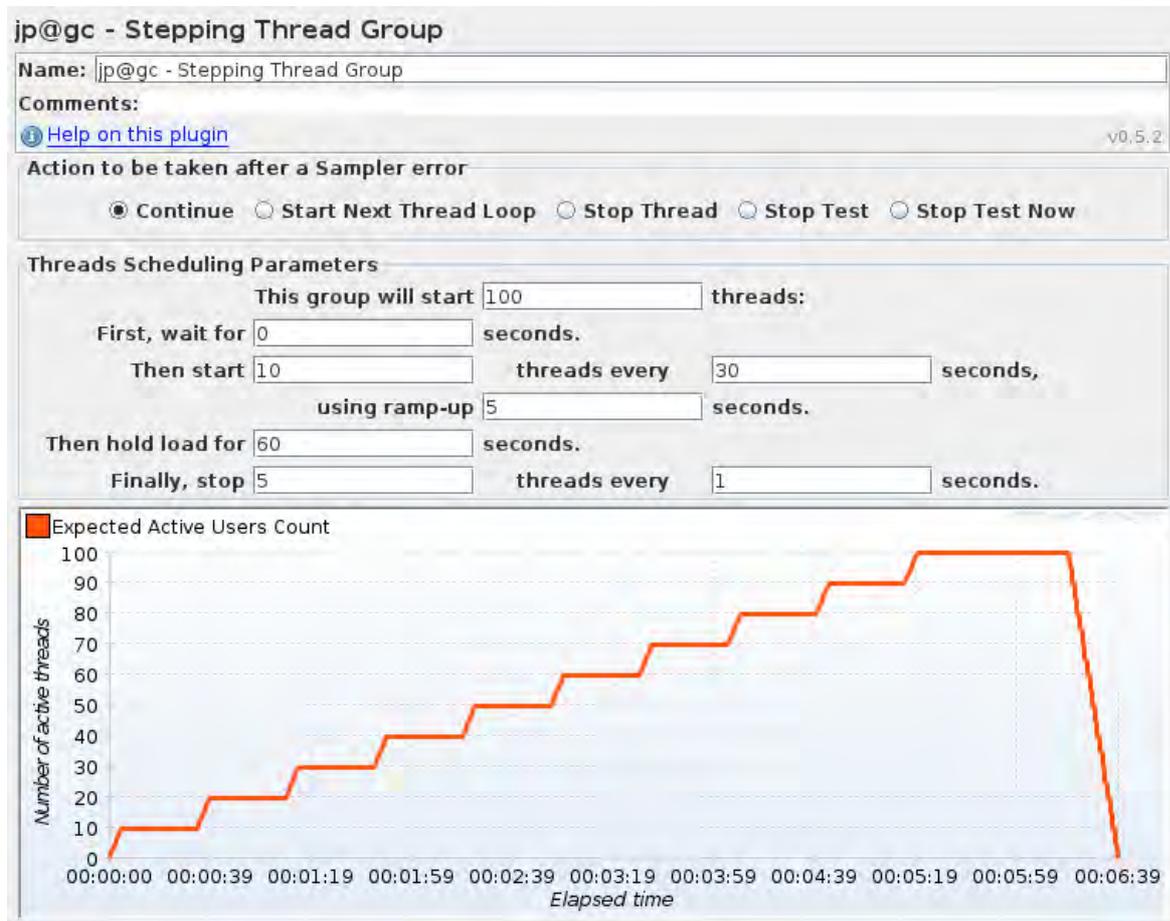


Figure 4 - Stepping Thread Group

Ultimate Thread Group (UTG) also offers threads schedule with preview. It is composed from ramp-up chunks and allows to compose complex cases, like sudden spikes. Every threads chunk has settings for startup delay, ramp-up time, hold time and ramp-down time. The final load will be calculated as a sum of the active threads count of all chunks in a given second.

Throughput Shaping Timer concentrates on request throughput and allows setting freeform schedule like UTG. It is used when your service is placed behind a load balancer and users count is masked.

Samplers

The original JMeter is perfect at testing HTTP tasks and other high-level TCP services. But it has no UDP protocol support. JP@GC offers *UDP Sampler*, which may be used for load testing widely used DNS servers.

Dummy Sampler is also interesting: it just produces samples without external requests, this helps to debug Regular Expression Extractor work.

HTTP Raw Request offers low-level TCP sampler with important improvement: memory consumption limits. It may be used for huge file uploads and huge response downloads when the original JMeter items produce `OutOfMemoryException` or work slowly.

Logging

As said above, experienced JMeter users like to run their tests from command-line using the non-GUI mode. The drawback is that JMeter shows no activity info in command line and it is difficult to tell if it is OK. Again, JP@GC helps with *Console Status Logger*, which prints short summary every second in non-GUI mode. Find it in the "Listeners" menu of JMeter.

A frequent question in JMeter Mailing List is how to save some value extracted during test to some custom log file. There is *Flexible File Writer* which can be used to setup freeform saving into file.

Inter-Thread Communication Features

Finally, the most advanced JMeter users will be surprised how easily they can perform the most complex task for JMeter scenario: pass some value from one Thread Group to another. The version 0.5.2 of JP@GC introduces Post-Processor that puts data into global queue and Pre-Processor that gets data from that queue, no matter in which thread it will be called.

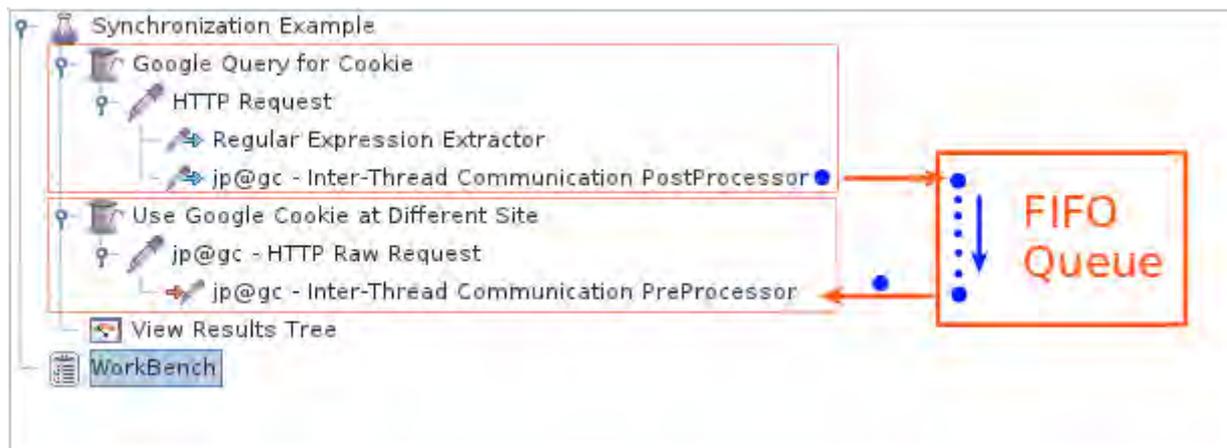


Figure 5 - Inter-Thread Communication via FIFO Queue

Conclusion

The original JMeter is a precious tool for the load testing community. And it can glitter even more when polished with JP@GC additions. Not all of them were covered in this short article, so take your time to investigate project documentation.

Remember, that you are in the area of free and open source software with JMeter and JP@GC. Your feedback and ideas can easily change the project and fill it with new features.

Introducing Practically Free Load Testing up to 1,000,000 Virtual Users

Its election season, and one of our clients got listed on the Drudge Report: a single page instantly got millions of hits before the site crashed. We thought it would be cool to provide an inexpensive load testing service that could test for this simple scenario so websites could make sure their site wouldn't be crushed when their page views go through the roof.

[Download Load Tester LITE from webperformance.com.](#)

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 60'000 visitors/month of our web sites! To advertise in this section or to place a page ad simply <http://www.methodsandtools.com/advertise.php>

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 © 2012, Martinig & Associates
