METHODS & TOOLS

Practical knowledge for the software developer, tester and project manager **ISSN 1661-402X**

Spring 2010 (Volume 18 - number 1) www.methodsandtools.com

Does Size Matter (in Software Development)?

When friends asked me what is the last trend in software development, I answered Lean. This approach is even easier to describe, because you can take examples outside the software industry like Toyota. Its recent problems shows that every idea could get difficulties nurturing its original values when scaling and software development is no exception to this rule. Agile has become "the thing to do" in software development and is now being used as the (marketing) label of every new initiative or tool. As a result, the fate of the original values of Agile Manifesto are to be diluted at best, abused at worst. I believe that the Agile Manifesto signatories were motivated by a sincere goal to give to the people involved in software development projects a better situation at a time when there could be a tendency to consider them as mere procedure performers. However, as their ideal spread and became successful, it meets the fact that software development is also a business for software tools vendors, consulting organizations... and media like Methods & Tools. Going from selling toaster to selling agile toaster could be now an mandatory move to be listed in the LeadingAnalystFirm Bermuda Triangle report and the front page of the press. It will however not bring any real benefits to agile or to toasters. A recent trade magazine report and tool vendor press release spoke about "taming the agile beast". This looks like a strange appreciation of Agile. Are thinking software developers dangerous animals? Does this mean that it is times to dump Mike Cohn and instead hire Siegfried and Roy to lead your projects? As Agile spreads, so are the chances that its initial ideas will be misunderstood... and that the number of failed projects claiming to follow the Agile approach will increase. I add the "claim" part, because some Agilists will reply that "true" Agile project cannot fail, but this would be the topic for another discussion. Sir Winston Churchill said "democracy is the worst form of government except all the others that have been tried.". The fact that it could be difficult to keep the ideal of approaches that rely strongly on participants' behavior when you scale them should not prevent us to aim for the best objectives. We have however to be realistic on the real world constraints, adapt to them and recognize that we cannot always reach perfection ;o) On this topic, I recommend the excellent books of Larman and Vodde on scaling lean and agile development. In the introduction of their first volume, they wrote: "Start with a small group of great people and only grow when it really starts to hurt". I could not give you a better advice. In our software development world, the "too big to fail" motto could easily be replaced by "too big to succeed".

[illegible]

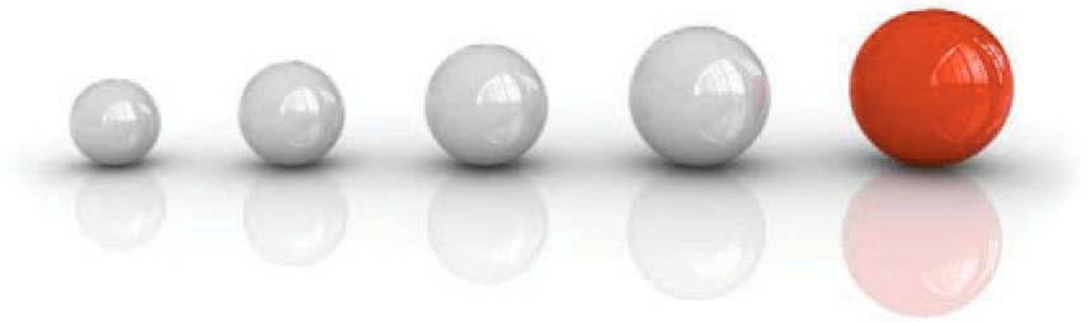
Inside

Looking for measurable value from Application Lifecycle Management?

• A major autmotive supplier cut a three year development cycle by a third.

• A utilities company accelerated release management processes by 80%.

• A global geo-content provider increased its On-Time measure from 70% to 93%.



You can achieve these kinds of results too.

When software is critical, you need a smarter approach.

You need Intelligent ALM™

1 800 613 7535 | On-demand Webinar:
www.mks.com/alm-webinar

MKS

Using WatiN to Leverage Common Elements in Web Testing

Ron Palmer, rpalmer @ exception13.com
Exception 13, Inc., www.exception13.com

What is WatiN?

WatiN is an easy to use, feature rich framework for **W**eb **A**pplication **T**esting **i**n .**N**et (W.A.T.I.N.). Inspired by and modeled after Watir, the framework has many benefits. It follows an open source model and the compiled libraries plus source can be freely downloaded, utilized and modified from the WatiN page on Sourceforge.net. Not only does it provide an extensible framework but also comes with the knowledge that over 60,000 users have downloaded the framework, making it one of the top 3 of open source web testing applications.

Being inspired by Watir, WatiN shares many of the same concepts and limitations as all Wati*X* ('*X*' being the language specific version, such as WatiN for .Net, Watir for Ruby or Watij for java) frameworks. It's important to remember that the '*X*' not only represents the development environment of the Wati*X* libraries but also the environment that will be use in the creation of the actual test code. This means that there is no secondary scripting language to master. You only need to be proficient in the language specific to your flavor of WatiX. It's also important to note that WatiN is browser dependent and only supports automation against Internet Explorer and Firefox. So if you are planning a cross browser testing effort with Safari or Opera requirements, you may want to look into other products.

Unlike many web testing platforms that use HTTP requests to emulate a browser transaction, WatiN operates at the browser object level. This allows for programmatic control of objects within a browser instance and gives the end user the ability to manipulate and exercise properties or methods exposed by the browser. This actual browser interaction allows for more direct testing of dynamic elements such as popup dialogs and AJAX controls. If you happen to be new to WatiN or Watir it is well worth the time to view the 'Getting Started' page noted in the **Further readings/downloads** section of this article.

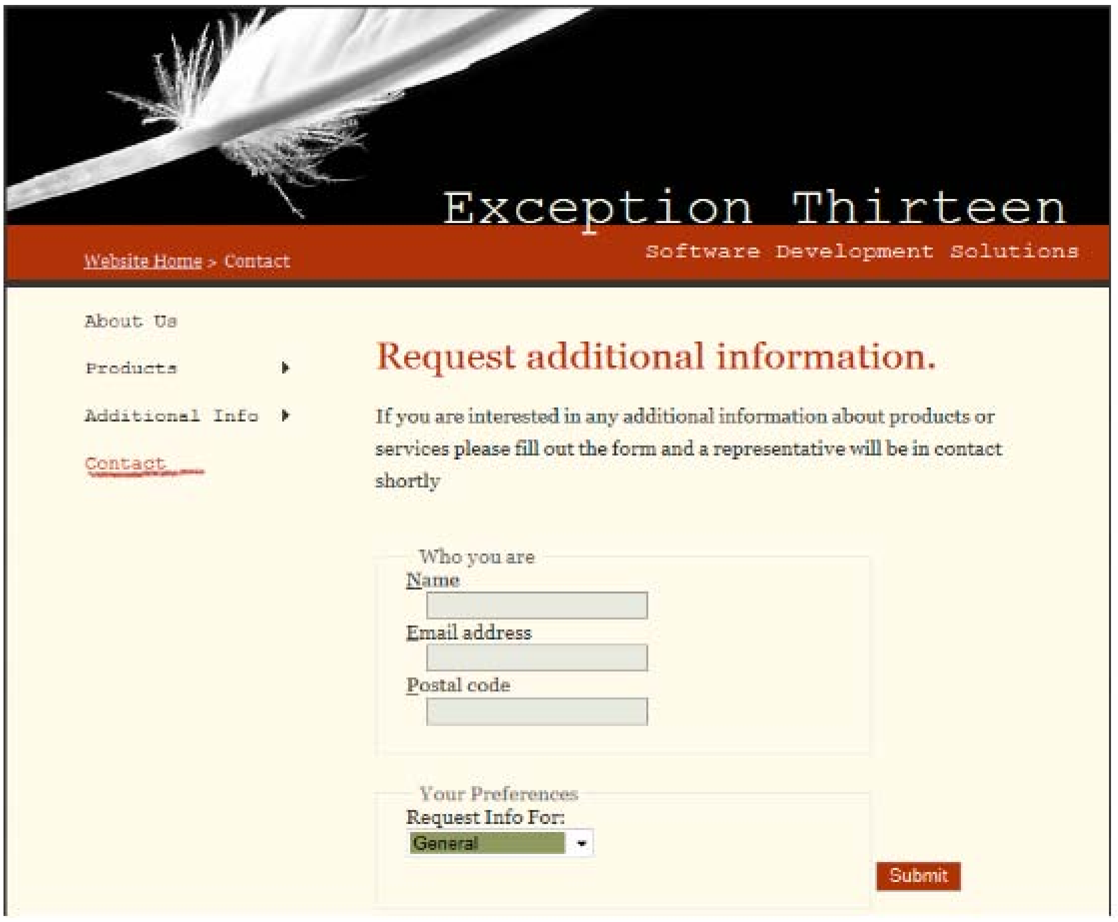
Even though WatiN is an outstanding framework, even the best frameworks are only as good as their implementations. This article will attempt to provide its reader with some practical insight into creating automation based on the WatiN framework. It will present a well thought out and proven foundation for general web testing. The final structure and format for your tests will vary based upon the unique requirements of your environment.

What are common elements?

There are many common elements on any given web page. These can be standard objects supported by a browser's Document Object Model or even custom controls provided by a third party. Regardless of the objects origin, most development shops treat common objects across pages in a similar manner. We can use this component of object oriented programming to create a framework that allows us to produce a library that supports objects and their functionality on a page.

Now you are probably asking yourself, "Great, so there are objects on a page. Now how are those objects classified as common elements for a library?" That right there is the billion dollar question (**Note:** *It used to be the million dollar question but I've adjusted the value based on inflationary measures and a falling dollar value*). Realistically, there is no one way to classify these objects. What matters is that your team defines a logical classification structure for the objects on a page and applied in a consistent manner across a project. In the following examples I will walk you through how I have classified common elements by page, type, and functionality for a simple “Contact” page.

A page is the highest classification of common elements within a site. This classification is a logical grouping of objects combined to make a web page. As you can see on the “Contact” page (Figure 1), there are several types of objects on this page. These include Images, labels, breadcrumbs, links, text fields, select boxes, and a button for this specific page. Keep in mind this is not meant to represent all objects that are possible on any page; just the objects that make up this page. When we build our library for this page you will notice that the page specific class is nothing more than a reference to each of the objects grouped together by its specific type.



Request additional information.

If you are interested in any additional information about products or services please fill out the form and a representative will be in contact shortly

Who you are
Name
Email address
Postal code

Your Preferences
Request Info For:
General

Figure 1. Contact Page

The object type classification is a more broadly defined classification based upon the object types within a site or project. Most projects have a defined GUI specification document or at least a general agreement within the development staff as to what objects are used within a site. This document normally will define acceptable colors, fonts, images, object types, etc. for use on any given page. Using this information we can define all of the object types expected to be used within a site. This will be the foundation for our object type library and will be leveraged as a common library for defining each page class.

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

Load Testing for all your web applications

NeoLoad

Test your web application's performance easily and deploy with confidence!

NeoLoad, **load testing solution**, ensures better efficiency to perform your tests faster, while providing pertinent analyses and full support for all new technologies.

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

- HTTP
- AJAX & RIA
- Adobe Flex
- Silverlight
- Oracle Forms
- SOAP
- GWT

More details and free trial on
www.neotys.com

NEOTYS
Make sure it works!

The final classification is by object functionality. Each object type has specific functional limitations. For example, a text box on a page may input text, delete text or display text. These are the functional actions associated to that object. Knowing the functional limitations and available actions of an object will allow us to create functions to support each of the functionality points.

How do I leverage these common elements?

Now that you have an idea of what WatiN is and some general concepts in classifying common elements within a page, we need to look at how to leverage these common elements. With our objects grouped in a logical manner, we can produce the code base to support the functionality of each object type. For the purposes of this article let's focus on a basic web component, the TextBox. When looking at a TextBox we can see that there are basic functional points that are covered by this object in the context of a site. For this site, we will create functions within a TextField class to support SetValue, GetValue, TextFieldExists and ValidateText (Code Sample 1). This will give us a basic level of coverage for functional testing of this page.

Code Sample: Test code for TextField.

```
namespace EX13.BaseElements
{
  public class TextField
  {
    private IE CurWin;
    private string BrowserTitle = "";

    public TextField(string _BrowserTitle)
    {
      BrowserTitle = _BrowserTitle;

      if (BrowserTitle == null)
      {
        Assert.Fail("No browser title was passed into BaseElements.");
      }
      CurWin = IE.AttachToIE(Find.ByTitle(BrowserTitle));
    }

    public void SetValue(object SE, string Value)
    {
      CurWin.TextField(Find.ById(new
Regex(SE.ToString()))).TypeText(Value);
    }

    public string GetValue(object SE)
    {
      string      TextValue      =      CurWin.TextField(Find.ById(new
Regex(SE.ToString()))).Value;
      Assert.IsNotNull(TextValue,    "Text.GetValue    FAILED:    '"    +
SE.ToString() + "' Returned a null value");
      return TextValue;
    }

    public void TextFieldExists(object SE, bool checkType)
    {
      if (checkType)
      {
       Assert.IsTrue(CurWin.TextField(new
Regex(SE.ToString())).Exists, SE.ToString() + " Does not Exist");
      }
      else
      {
       Assert.IsFalse(CurWin.TextField(new
Regex(SE.ToString())).Exists, SE.ToString() + " Shouldn't exist");
      }
    }

    public void ValidateText(object SE, string Value)
    {
      Assert.AreEqual(CurWin.TextField(new
Regex(SE.ToString())).Value, Value);
    }
  }
```

By creating a class to define functions for each functional point of an object type, we create an abstraction layer that allows us the ability to combine common IDE elements like Asserts with the core functionality of WatiN. This library assures that a test method utilizing this class will have a standard set of supported functionality and consistent handling of object specific logic. Not only does this reduce code within the test methods and Page Oriented Libraries but it also reduces implementation complexity.

With the Object Type functionality library built, we can now focus on building Page oriented libraries. The Page oriented libraries are used to define the objects that exist on a given page. We will create a specific class for each page that defines the named objects on that page using enums to list each object by type (Code Sample 2). Each enum value will contain the ids of objects on a page based upon its type classification. For example the page contains three input fields with the ids ‘TextName’, ‘TextEmail’, and ‘TextPostal’. These will be public members of the ContactPage class and will be available to each test method that utilizes the class.

Code Sample: Test code for ContactPage Class.

```
namespace EX13.Page
  public class ContactPage
  {
    public enum EnumTextFld
    {
      TextName,
      TextEmail,
      TextPostal
    };

    public enum EnumSelect
    {
      SelectReqInfo,
    };

    public enum EnumLinks
    {
      LinkDefault,
      LinkAbout,
      LinkProducts,
      LinkAdditionalInfo,
      LinkSubmit
    };
    public enum EnumLabels
    {
      LabelWhoYouAre,
      LabelName,
      LabelEmail,
      LabelPostalCode
    };

    public EX13.BaseElements.TextField textfield = null;
    public EX13.BaseElements.SelectList selectList = null;
    public EX13.BaseElements.RadioButton radioButton = null;
    public EX13.BaseElements.Link link = null;
    public EX13.BaseElements.Element element = null;
    public EX13.BaseElements.CheckBox checkBox = null;
    public EX13.BaseElements.Label label = null;

    IE _searchWindow = null;

    public ContactPage(IE searchWindow, string screenName)
    {
      _searchWindow = searchWindow;
      textfield = new EX13.BaseElements.TextField(screenName);
      selectList = new EX13.BaseElements.SelectList(screenName);
      link = new EX13.BaseElements.Link(screenName);
      label = new EX13.BaseElements.Label(screenName);
    }
  }
}
```

Leverage the page Libraries to create tests.

Once the Page oriented classes are defined we can begin to leverage these classes with the Object Type functionality to produce well defined test methods. This process is fairly straightforward and provides the tester who is responsible for the creation of the test a finite choice of available objects to write test code against and decouples the test method from the actual page under test. This decoupling reduces the impact of object property and functionality changes within a site.

Web Performance Testing: Simple, Fast, Realistic, Reliable, Precise
Multiple Application Modes From One Test Script:
Functional Testing with AJAX/Web 2.0 Support
RIA Monitoring with Easy Reporting Integration
Multiple Parallel Playbacks for Server Loading
eValid
www.e-valid.com
Analyze Your Web Applications from the Real End-User Perspective
Free Evaluation Download

Code Sample: Test Code for ContactPage Tests.

```
[TestMethod]
    public void Assert_ContactPage_Objects()
    {
      var ContactWindow = new IE("http://www.test.com/contact.aspx");
      Contact    ContactScreen    =    new    ContactPage(ContactWindow,
"ContactPage");

      //Validate List Boxes Exists

ContactScreen.selectList.SelectBoxExists(Contact.EnumSelect.SelectReqI
nfo, true);

      //Validate Labels Exists and text

ContactScreen.label.ValidateLabelText(ContactPage.EnumLabels.LabelWhoY
ouAre,
      "Who you are");

ContactScreen.label.ValidateLabelText(ContactPage.EnumLabels.LableName
, "Name");

ContactScreen.label.ValidateLabelText(Search.EnumLabels.LabelEmail,
      "Email address");

ContactScreen.label.ValidateLabelText(Search.EnumLabels.LabelPostalCod
e,
      "Postal code");

     //Validate Text boxes Exists

ContactScreen.textfield.TextFieldExists(ContactPage.EnumTextFld.TextNa
me, true);

ContactScreen.textfield.TextFieldExists(ContactPage.EnumTextFld.TextEm
ail, true);

ContactScreen.textfield.TextFieldExists(ContactPage.EnumTextFld.TextPo
stal, true);

      //Validate Buttons and Links
      ContactScreen.link.LinkExists(ContactPage.EnumLinks.LinkDefault,
true);
      ContactScreen.link.LinkExists(ContactPage.EnumLinks.LinkAbout,
true);

ContactScreen.link.LinkExists(ContactPage.EnumLinks.LinkProducts,
true);

ContactScreen.link.LinkExists(ContactPage.EnumLinks.LinkAdditionalInfo
, true);
      ContactScreen.link.LinkExists(ContactPage.EnumLinks.LinkSubmit,
true);
     }
```

All of the source code displayed in this article was produced using Microsoft Visual Studio Team System (VSTS) 2008 Test Edition. If you currently do not have VSTS installed, a trial copy can be downloaded from Microsoft.com (**Mandatory disclaimer**: *The author of this article is not an MS ideologue… Really, I swear. Some of my best friends work on Java projects. I just happened to be working on a .Net project during the writing of this.*). If you happen to be using another language, IDE or automation framework such as Selenium IDE or Watir, the concepts are general enough to apply to any development environment with slight modifications for syntax. I have a firm belief (call it a dream if you will) that an automation framework should not be judged by the color of its IDE but the content of its characters.

Further readings/downloads

Product	**URL**
WatiN	http://sourceforge.net/projects/watin/files/
WatiN - Getting Started	http://watin.sourceforge.net/gettingstarted.html
Watir	http://rubyforge.org/projects/wtr/
Selenium IDE	http://seleniumhq.org/download
Microsoft VSTS	http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=d95598d7-aa6e-4f24-82e3-81570c5384cb

Advertisement - SpiraTest - End The Testing Chaos! - Click on ad to reach advertiser web site

Used by over 600 customers worldwide...
SpiraTest® - End The Testing Chaos!
Why spend money on standalone requirements management, bug tracking and testing tools?
SpiraTest® provides a complete Quality Assurance solution that manages your Requirements, Test Cases, Test Sets, Bugs and Issues in one environment with complete traceability from inception to completion.
spiraTest®
Version 2.0 Professional Edition
inflectra
> Learn More | Download Free Trial

Five Symptoms of Mechanical Agile

Daryl Kulak
Pillar Technology, http://www.pillartechnology.com

Although I'm an Agile coach by trade, I'm really a storyteller. In this article, I'd like to tell you five stories, mostly based on real life, that might help you see how Agile can become *mechanical*. I have changed the names to something ridiculous to make sure I don't implicate anyone.

Agile Expert Syndrome

Once upon a time, in a company called Neverland Insurance, there was a team and an Agile consultant. Together, they decided to try to build an application. The team was about ten people, and the Agile consultant had a really, really good reputation as a brilliant person and dedicated Agilist. The team felt lucky to have him. The Agile consultant's name was Sticky LaGrange.

Sticky helped the team get started and taught them all the Agile practices. The team carefully followed Sticky's words and asked him questions whenever they weren't exactly sure what to do. Sticky was very helpful and patient. Not only that, but Sticky prided himself on writing more code than any other team member, even though he was spending so much time coaching people. Sticky was truly amazing.

At the end, the application was finished on-time and within budget. Everyone at Neverland was so impressed. The team was so proud.

Then a wicked king of Neverland cut budgets and the team had to say goodbye to Sticky. Surely they could they succeed without Sticky, he had taught them so well. So the team went on to another application and began work. They tried to remember everything that Sticky had told them. If in doubt, someone in the team room would inevitably ask "What would Sticky do?" The group would try to remember what Sticky would have done and then replicated it faithfully. But things were falling apart. Different team members remembered Sticky's advice differently. And sometimes, inexplicably, Sticky's advice seemed to be wrong! When applied in this new situation, Sticky's advice broke down. How could Sticky be wrong?

Some team members said that maybe they should go beyond Sticky's advice. But other people thought this was a terrible idea. "That's not True Agile!" they cried. Why deviate from Sticky's advice if it had worked so well the first time?

Unfortunately, the team at Neverland had fallen into what I call the **Agile Expert Syndrome**. This is when people feel like they *have to* follow the advice of an expert, either in the form of a consultant or a book or an article.

Whenever a team gets into this compliance mode, they are in trouble. They are performing Agile *mechanically*. And teams that are doing Mechanical Agile always get into trouble, because they will always hit a snag that falls in between the advice they received from the Expert. And as soon as that happens, they will get into a rut where they are trying to guess what the Expert would do, instead of applying their own intelligence and doing the best thing they can think to do.

Advertisement - SPA Conference - Click on ad to reach advertiser web site

Spa
2010

Software Practice Advancement 2010

Technology... Practice... Process... People... Participation

16 - 19 May 2010

BCS London (Covent Garden)

SPA2010 brings together experts and practitioners across a variety of industries, from around the world, to share the latest thinking in software development. At the forefront of innovation in software since it's inception, SPA2010 highlights the advances in technology and practice that make a real difference to your skills, to the effectiveness of your teams, and to the quality of the software you produce.

SPA2010 sponsors

Cincom.

RALLY SOFTWARE

ThoughtWorks

zühlke empowering ideas

bcs The Chartered Institute for IT

To book a place or find out more visit www.spaconference.org or call +44 (0)115 860 2009

Separation of Decision-Making and the Work

Once upon a time there was a clothing manufacturing company called Cheap Green Shirts.com. The company had several Agile teams that were doing a great job and were highly productive. Their velocity was high, the teams were happy and the business was benefiting.

“I think we can juice these teams up even more,” said one of the vice-presidents. “I’ve noticed that Team A is doing significantly better than the other teams, so we should take the best practices from Team A and give them to the other teams. That way, we’ll have all teams performing at the level of Team A. It will be great!”

So the vice-president set up a group of Agile experts that he called a Software Engineering Process Group (SEPG) that was able to advice the teams. The SEPG (pronounced seepage) experts carefully examined each practice used by each team, harvested them and chose the best of the best, then gave them back to the teams.

Surprisingly, the teams didn’t use many of the new practices. The SEPG experts were incensed. “These teams don’t want to learn,” the SEPG experts groaned. “The SEPG dudes don’t understand our work,” said the teams.

So the vice-president mandated that the groups use what the SEPG experts had come up with. The groups dutifully used the new practices. At this point, velocity dropped for each and every group. How could this happen? The SEPG experts explained “The teams are doing the practices wrong!” The teams replied “These practices don’t work for us.”

The vice-president fell into a mechanical Agile problem that we call “separation of decision-making and the work.” The further the decision-making happens away from the person doing the work, the worse the result will be. The SEPG group was far removed from the teams, they weren’t familiar with the details of the day-to-day work, and so, even though they were experts, their recommendations failed miserably. At the beginning, the teams were making their own decisions about which practices to use, which ones were best for them. By removing that decision-making from the local team members and giving it to a separate group, the vice-president reduced velocity, team satisfaction and caused problems for everyone. The vice-president was thinking that the teams were machines, not people.

Blame the Person, Not the System

A company had two teams working together. One team was providing a set of requirements for the other team to implement in code. The first team we’ll call the Uppities, because they were upstream in the process. The second team we’ll call the Downers, because they were downstream in the process.

First of all, the Uppities and the Downers were both doing Agile, just different variations. The Uppities were on a longer iteration cycle of 6 weeks, whereas the Downers were using 2 week iterations. But everyone was doing some flavor of Agile.

Still, the Uppities did not get along too well with the Downers. The Uppities tended to have big emergencies and would dump high-priority work into the Downers area, even in mid-iteration. Since the Uppities work was the most important work in the company, the Downers had to comply.

One day, I was in a conversation with someone from the downstream team, the Downers. Let's call him Donny.

I asked Donny what he thought was the origin of the conflicts between the Uppities and the Downers. “Well, it's the Uppities, of course!” he said. “They have no respect for other teams! They think they own the place!”

I asked Donny if he could imagine what it was like from the Uppities perspective.

“Oh sure, that's no problem. I used to work there,” Donny said, to my surprise. “And when I was an Uppity, it was pretty obvious that the Downers were the problem. But now I can see the REAL story, that the Uppities are the entire problem, and not the Downers!”

I couldn't help wondering if Donny was falling into another aspect of Mechanical Agile, which I call “Blame the Person, not the System.” Donny, no matter which team he belonged to, was thinking that the other team was the problem, instead of thinking that the system was the problem. By system I don't mean the software application, I mean the methods of interaction between the two teams, even the culture. There were probably problems in the overall system that were causing this conflict, and those systemic problems were being misinterpreted as “the other guy's fault.”

If we can always take an attitude of “Let's fix the system together” rather than getting into a blame-game of determining who's at fault (Hint: it's always the other guy), we can make much more progress.

Just Tell Me What to Do, Boss

My employer, Pillar Technology, has a very rigorous interview process for developers. Part of our process is that the prospective recruit has to pair with one of our Pillar experts to allow us to evaluate the person's ability to perform as a test-driven developer.

In one case that I was able to observe from across the team room, a prospective recruit came into the room and sat down with our Pillar expert. We'll call the prospective recruit Emo. Since this was about test-driven development, the Pillar expert asked Emo to look at some existing code, and an existing unit test, and try to figure out how to make the test pass.

The test was very simple. It provided some input and then asserted that the variable being returned should equal “12.” Emo looked at the test, then flipped over to the code. After a quick glance at the code, Emo took the cursor and deleted it all. In its place, Emo wrote code that would make the return variable equal to “12.” Then Emo smiled at the Pillar expert.

This was both the shortest pairing interview I have ever seen, and the longest post-interview retrospective.

Emo fell into an aspect of Mechanical Agile that I call “Just Tell Me What to Do, Boss.” If our team members get into a mindset where they just want to take direction from their boss, we're all in trouble. We need every team member to be applying his or her full intelligence to every problem. As soon as they get into the mode of following directions, then we get solutions like what Emo did in the interview.

Team members can get into this way of thinking quite easily. Managers who are coercive create team members like this, because people are afraid to do anything different than exactly what the manager specified. But even the presence of detailed documentation on “what you should do” can create these types of team members.

We need to do everything we can to avoid getting our team members into a mindset like poor Emo.

Competition Between People and Teams

A VP at a telecom company had six Agile teams all doing great work. Then he had an idea. He realized he could “kick it up a notch” by getting his teams to compete against each other for a monthly prize.

We’ll call this VP Emeril.

Emeril said he would offer free lunches for a week for the team that could top the other teams in velocity.

The competition began. The teams immediately started looking for ways to game the system. They found that if they made their storypoints smaller, they could deliver more of them without doing extra work. The teams went through a period of “storypoint inflation” for several months until the VP realized what was going on. Finally, he mandated that the storypoints had to be all the same size.

Advertisement - Agile 2010 Conference - Click on ad to reach advertiser web site

Then the teams realized that quality and defects were not part of the measurement, so they reduced the amount of time they spent on testing in each iteration. Sure enough, the points went up, but so did the defects. So, the VP eventually jumped in and mandated that defects would be part of the measurement. The more defects found, the lower the team would score.

This sounded okay, but the VP didn't realize that the testers were also part of the teams. They were also in line for free lunches. So, by making this third decree, the VP was telling the testers "Don't find too many defects or you won't get the free lunches."

And on it went. You can imagine that these competitions didn't help things much. And the cooperation that naturally was occurring between the teams disappeared. Why help the other team gain velocity at our expense?

The VP was suffering from a symptom of Mechanical Agile called **"Competition Between People and Teams."** Put people or teams in competition with each other and you will ensure that they will meet your targets, even if they have to destroy your organization to do it.

Five Symptoms – Five Stories

These are the problems that keep your Agile teams from scaling up and sustaining. And they all relate back to treating people like machines instead of like people.

To solve the Mechanical Agile, no amount of new practices or Lean, Kanban or Six Sigma will help you. Scrum of Scrums won't really help you. More meetings won't help you. Only addressing the attitude of thinking of people as machines will help you.

Some Ideas That May Help

Here are some suggestions to improve your teams' ability to scale and sustain. These aren't meant to be singular solutions to the first five symptoms, you could say that all five of these solutions solve all five symptoms.

- If You See a Best Practice By the Side of the Road – Kill It

The idea of "best practices" can become quite mechanical. It can lead directly to Agile Expert Syndrome of any of the problems we've discussed above.

I'm not saying that you can never take a practice that you see elsewhere and try it in your situation. I'm saying that *you shouldn't do it blindly*.

Blindly following "best practices" always gets us into trouble. We can try to understand practices used elsewhere, but then we can jump into answering "How can that work here? How do we need to change it to fit?"

- Close the Gap Between Decision-Making and the Work

As we saw with the SEPG experts, it hurts when we take decision-making away from the person doing the work. The further that gap, the bigger the hurt. We need to move decision-making as close to the work as possible. If the person doing the work can make all decisions about that work, that is optimal. If we have to separate it for some reason, try to keep it as close as possible (within the same team, hopefully).

- Break Down the Boundaries Between Teams

A manager's biggest responsibility is to break down the boundaries between teams. Having teams compete against one another erects boundaries. A manager stating that "Everything must go through me" to communicate to other teams erects boundaries.

The best Agile manager will work to remove boundaries, finding ways the team can have unfettered communication with other teams, groups, departments.

- Value the Unstructured

Quite often when we see a problem, we try to invent some structure to fix it. Have two Agile teams that need to communicate more? Set up a Scrum of Scrums meeting.

But there is also value in the unstructured. Sometimes it is worth trying to find a way to solve problems that doesn't involve more structure, more meetings, more roles, more documents, more setup.

Are there things we can change about our attitude that can improve the situation? If so, what can we do to change our own attitudes? How can we change the culture of the way work is done in our team?

- Avoid Over-Engineering Your Processes

We often use the term software engineering to describe how we create software. Software, once created, is certainly mechanical, so thinking of ourselves as building something mechanical can make sense when describing software creation.

However, it can cause trouble when we try to engineer processes. Our processes cannot be mechanical, for all the reasons in my stories above. They have to be living, breathing processes that can adapt effortlessly. The only way to have processes that are alive like this are to have openings in them for people to do what they think is the right thing to do. We need to allow people to act like people, and not try to force them into a machine model that we've created for them.

Advertisement - MKS Enterprise Agile Solution - Click on ad to reach advertiser web site

A bigger whiteboard isn't the answer for scaling Agile.
Choose MKS Integrity to power your enterprise Agile transformation.
Get the whitepaper:
Agile Development Doesn't Have to Mean Fragile Enterprise Process
http://www.mks.com/enterprise-agile
MKS

Writing Testable Code

Isa Goksu, info @ isagoksu.com,
ThoughtWorks Inc, http://www.thoughtworks.com

Every domain has its own best practices and disciplines. Let's think about an English teacher. He has to follow certain practices and adopt certain disciplines to be successful in his job. He cannot sing a song, or use the sign language to give his lecture at its best. There are certain methods and certain practices to follow. These practices come from different experiences, people and time. I'm not going to explain how an English teacher should teach, I'm sure there are many books about that. My point was about “the existence” of different methodologies, practices and disciplines to follow. These practices tell you not only how, they also tell you what, when, where and whom. This doesn't mean that you cannot create your own methodology. However, it's a fact that many people who tried to create new methodologies have failed.

I think programming is the art of translating business requirements into a computer language. And just like teaching, programming has its own practices to follow and disciplines to adopt. Test driven design, pair-programming, code-reviewing are just couple of them. There are many books that explain all these practices/disciplines. However, in this article, I would like to focus on programming from a different aspect: “Writing Testable Code”. To me, this little concept is not getting enough attention. And I think it definitely needs more attention.

Problems of Dirty Code Base

Developing new software isn't just about to accomplish the business requirements within the given time frame. You have to think about the maintenance and support of the created software as well. So what happens when we write dirty code? Basically you end up with a mess. A mess which adding a new feature, changing the behavior of a component, replacing UI elements with the new ones, or any maintenance/upgrades that you could think of would be very hard.

Please notice that this problem has nothing to do with you being agile or your design skills. This is about whether someone else can understand your code after development or not. It's about making yourself indispensable or replaceable within the team, or how easy it is to introduce someone new into the team.

Isn't Unit Testing Enough?

Short answer is "No". Unfortunately we tend to think that if we have unit tests, it solves all the problems. Well, this is obviously not true. Unit tests are tools that help you to test your code whether it behaves the way you want or not. It does nothing else.

In our case, yes unit tests might help understanding the code base. However, the quality of the tests and the total test coverage in the code base is often case not good enough. It is very likely to end up having only 45-55% test coverage and 65-70% quality in the unit tests.

Besides, having unit tests doesn't prove that you followed TDD (Test Driven Design) practices. So by looking at this picture, we can presume that "Unit Testing" is actually not enough.

Advertisement - DevDay for iPhone Conferences - Click on ad to reach advertiser web site

London: April 26, 2010 & Geneva: April 28, 2010

DEVELOPER DAYS

DevDay for iPhone

DevDay for iPhone is a dedicated and interactive one day event for software developers and business professionals who want to learn how to successfully build and market iPhone applications.

win an iPad 16GB

TOPICS

Everything you need to know about launching an iPhone App

Core Data
iPad*
Agile iPhone development
Case studies

Best practices of launching iPhone applications

#Building animated iPhone user interfaces

SPEAKERS :

Daniel Steinberg
Adrian Kosmaczewski
Raven Zachary
Sumit Rai
KC MacLaren
Neelan Choksi
Frederic Layani
Jérôme Layat

*subject to the release of the current NDA on the iPad SDK by Apple

DATE LONDON: APRIL 26, 2010
Dexter House
No.2 Royal Mint Court
Tower Hill London, EC3N 4QN
Price London: £399 + VAT
www.iphonedevday.com/iphone-london-2010/

DATE GENEVA: APRIL 28, 2010
Geneva Business Center
12, Avenue des Morgines
CH-1213 Petit-Lancy
Price Geneva : CHF 450
www.iphonedevday.com/iphone-geneva-2010/

DevDay for iPhone is an independent event that has not been authorised, sponsored or otherwise approved by Apple Inc., iPhone is a trade mark of Apple Inc., registered in the EU and other countries.

Unit Testing vs. Test Driven Design

Unfortunately many developers still don't know or don't think about the difference. As I mentioned earlier, “Unit Testing” is the practice of testing your own code to understand whether it behaves the way you want or not. On the other hand Test Driven Design is the discipline of using “Unit Testing” when you design the software. This small difference actually makes big impact on your software design. Let me explain why:

If your development team is using “Unit Testing”; creating the code base first (even a single unit) and write the tests after creating the code base, it actually means that you’re hanging yourself with “Unit Testing” rope. The reason is every time you need a change in the design that affects your unit; you will have to fix the existing unit tests too. In the beginning it seems like a small problem, but it will be a real big problem later in the development phase since you will have lots of unit/integration tests depend on certain design assumptions. And when this design changes, your tests will all have to be changed. After some time you will also notice that your development team is spending more time on fixing tests than developing the software itself. Moreover, it is very hard to identify “what to test” for an already written code than a code that you are about to write. This is also why many developers that follow “Unit Testing” start testing the methods in the unit than the unit behaviors.

However, if your development team is adopting the TDD discipline, you will notice all these design changes will be identified in the early phase [1] by developers. Writing tests first will allow you to discover the spots where new design is crucial, just by showing how hard/easy to write the test for that particular behavior.

On top of this, TDD will ensure that your unit/integration tests will be based on behaviors rather than methods. As we all know, most design changes are just the separation of concerns within the unit/module. Another way of saying this is, your responsibilities will shift to some other unit/module, but the behavior will still exist in your system. As you can see, instead of maintaining the existing unit/integration tests, you will be just moving them to the new unit.

I guess this difference is a subject for another article.

What Problem Exactly Are We Talking About?

Robert C. Martin has a really nice book about “Clean Code”. This book covers lots of hints about agile software development. I strongly recommend you to read it. I want to quote something from the book:

“You are reading this book for two reasons. First, you are a programmer. Second, you want to be a better programmer. Good. We need better programmers.” [2]

What is better programmer? I believe Uncle Bob is not talking about the programmers who can write one-liners (one line that does the whole behavior). Once you read the book, you will understand that he is actually talking about the programmers that can write clean, easy to understand code. I want to add one more thing to this, which was not in the book. Better programmers are always replaceable in the team.

So how do we ensure that our development team consists of better programmers? Well, this is a topic for a book unfortunately. However, I believe those are the programmers who can manage to write the simplest thing that could possibly work [3], and they are replaceable anytime within the team.

Let’s look at the possible problems within a development team that are the symptoms of dirty code base:

- When you introduce a new team member, it takes more than a day for him to start coding
- When your team members are becoming more indispensable every new day
- When you want to change the behavior of newly written code piece, it requires original developer to do it
- When your team members are spending too much time on testing
- When nobody could understand certain code pieces without rewriting them
- When your team members don’t want to work on certain area
- When senior team members having hard time to mentor new team members

Yes, all these are the symptoms of dirty code base. Why is this dirty code? It is a dirty code because all these problems are the results of some concessions while coding. Concessions that make your code hard to understand, follow, change or replace. And when you make some concessions, next developer makes his own concession on top yours. So what are these concessions then?

1. Adding too many dependencies (create big object graphs)
2. Hiding dependencies (i.e. within constructor, private/static methods, etc)
3. Making objects hard to initialize
4. Creating so many execution paths from a single point
5. Using train wrecks (trying to use/access a method/property by chaining so many other objects)
6. Using too many arguments for a single method
7. Creating longer methods/classes
8. Breaking SOLID [4] rules
9. Repeating yourself
10. Over-designing/thinking
11. And many more…

In the next chapter, I will try to address some of these problems and solution proposals. I believe if we try to make our code testable, we can take a good big step towards being a “Better Programmer” and having a clean code base. And I don’t think these are really hard things to do. Most of them are pretty simple and easy to accomplish by just paying a little more attention to details.

Testability

I would like to mention that this part of the article is not about software design. This is all about paying more attention to details and focusing on testability. All these items can be controlled while doing a code review.

Working with new code base and existing code base has different problems when it comes to testability.

Advertisement - Jazoon Java Conference - Click on ad to reach advertiser web site

JAZOON.COM

WHERE JAVA PEOPLE MEET

Keynote Speakers

- Kevlin Henney
- Ken Schwaber
- Hans-Willem van Vliet

JAZOON

THE INTERNATIONAL **CONFERENCE** ON **JAVA** TECHNOLOGY
1-3 JUNE 2010, **ZURICH**

AdNovum canoo COMIT CREDIT SUISSE netcetera ORACLE zühlke

For new code base, you can pretty much apply all agile and XP (eXtreme Programming) practices. On the other hand applying these practices to existing code base is not always possible. In this article I would like to focus on new code base only. However, I would like to write a separate article for existing code base and testability.

New code base is always fun to work with. As I mentioned earlier, you can pretty much apply all agile and XP practices. I will assume that you are already familiar with these practices. If you are not, you can still continue reading. However, I would definitely recommend you to look at these practices [5] in no time.

Too Many Dependencies

First rule of writing testable code to me is having a good dependency management within the code base, or at least having the least possible dependencies in your object/module relations. Basically, when you refer another object within your object, you depend on that object. When these object dependencies reach to a certain level, you would have unbreakable code pieces. Or in other terms, it would be impossible to understand this particular code piece without understanding lots of other code pieces. And sometimes this goes to deep and you would have transitive dependencies; you depend on a unit, and that unit depends on some other units. And this big chain of dependencies brings so much information overhead that nobody can easily understand the unit/module after a while.

Every dependency ties your unit to another unit/module. It means without having that unit/module, you cannot use your unit anywhere in the system. However, having no dependency is very theoretical situation. When you use database access, you would probably depend on the database layer in the application. When you modify the customer details, you would probably depend on the customer service in the application.

So is this a problem? In fact, it is not a problem. Having a centralized database layer or a customer service is always a good idea. However, if your unit has more than 3-4 of these dependencies, then it is a big problem. It means that, you are introducing more possible change reasons to your code. According to “Single Responsibility Principle” [6], your unit cannot have more than one reason to change.

Another aspect of this problem is the unit tests. When you have too many dependencies, you will find yourself trying very hard to instantiate your unit while writing the unit tests. Because you have too many dependencies, it will be very hard to instantiate the unit under test. Let me try to explain this with some code snippets:

```
class CreditCard {
    PaymentService paymentService;
    CustomerService customerService;
    OrderService orderService;
    EmailService emailService;
    // more dependencies...

    void charge(amount) { /* use these services... */ }
}
```

And a possible unit test for charging:

```
testChargeNothingWhenNegativeAmountIsGiven() {
  CreditCard card = new CreditCard();
  PaymentGateway paymentGateway = new PaymentGateway();
  PaymentService paymentService = new PaymentService();
  paymentService.setPaymentGateway(paymentGateway);
  CustomerService customerService = new CustomerService();
  OrderService orderService = new OrderService();
  EmailService emailService = new EmailService();

  card.setPaymentService(paymentService);
  card.setCustomerService(customerService);
  card.setOrderService(orderService);
  card.setEmailService(emailService);
  // many more setups
  // and your test case
}
```

As you see, after 3-4 dependencies your code starts becoming unreadable. So how do we solve this problem? Actually it’s very simple. Keep the dependent units that have same reason to change together and the rest separate. In above example, Email Service has definitely different reasons to change than our unit. On the other hand Payment Service, Order Service and Customer Service have more likely to have same reason to change. So separating different concerns out of our unit would make it cleaner, and easier to understand. And more importantly, it will reduce the dependencies of our unit.

To solidify above example, creating an asynchronous Email Job that uses the Email Service to take care of the emailing part will reduce the dependencies. And/or replacing the logic that uses Customer/Order Service to find out some information about the Customer/Order with the information itself that Credit Card unit needs, will definitely reduce the dependencies.

If we can pay a little more attention to this little dependency chain, our code will be clearer to understand.

Hidden Dependencies

Another problem that comes with dependencies is hidden dependencies. Hidden dependency is a dependency that you cannot see the dependency through the code easily and it is generally noticed in the run-time only. Let me give you an example of it:

```
// .. some code
creditCard.charge();
// .. some other code
```

When we run above code, we get an exception in the run-time that says: “Payment Service is null/not assigned”. Then we realize that we forgot to set the payment service. And we set it:

```
// .. some code
PaymentService paymentService = new PaymentService();
creditCard.setPaymentService(paymentService);
creditCard.charge();
// .. some other code
```

When we try second time running above code, we get another exception in the run-time that says: “Payment Gateway is null/not assigned”. Then again, we realize that we forgot to set the Payment Gateway for the Payment Service. And we set it too:

```
// .. some code
PaymentGateway paymentGateway = new PaymentGateway();
PaymentService paymentService = new PaymentService();
paymentService.setPaymentGateway(paymentGateway);
creditCard.setPaymentService(paymentService);
creditCard.charge();
// .. some other code
```

When we try one last time running above code, we get some other exception in the run-time that says: “PayPal Web Service is null/not assigned”. After figuring out all the dependencies and set them properly, we can run our code without a problem:

```
// .. some code
PayPalWebService paypalWebService = new PayPalWebService();
PaymentGateway paymentGateway = new PaymentGateway();
paymentGateway.setClearingHouse(paypalWebService);
PaymentService paymentService = new PaymentService();
paymentService.setPaymentGateway(paymentGateway);
creditCard.setPaymentService(paymentService);
creditCard.charge() ;
// .. some other code
```

As you see, it takes 2-3 tries to find out all the minimum dependency settings for our unit. And the process of identifying these dependencies won't make a big change for even an experienced developer.

You can think that having a dependency injection framework solves all these problems. Yes, it will indeed solve our problem here. However, it won't make your code testable. When you try to test above piece of code, you will still need your dependency injection framework which is itself another dependency. Isolating your unit will be almost impossible. Besides, understanding the code is still another aspect of this problem. Even though you have a dependency injection framework, it doesn't mean that you code is easy to understand.

So how do we solve this problem? Well, making your dependencies obvious. To do so, we can use one of several alternative methods. My favorite is using Constructor Injection. Constructor Injection is a method of injecting your dependencies to your unit in the construction time. Let's see in an example:

```
class CreditCard {
    private PaymentService paymentService;

    public CreditCard(PaymentService paymentService) {
        this.paymentService = paymentService;
    }

    // more code..
}
```

If we try to use this unit anywhere in the system, it is very obvious that this unit requires a Payment Service to work with. And when you try to test the unit, it is now very easy to inject a fake/stub/mock Payment Service which doesn't require any initialization.

Another possible solution to this problem is, using method level injection. Again, the idea stays same: make your dependencies obvious.

```
// some code..
public void charge(PaymentService paymentService) {
    // use this service
}

// more code..
```

Dependency Lookups

Another dependency related problem is Dependency Lookups. Dependency Lookup is a technique that comes from several patterns like Service Locator Pattern, Repository Pattern, etc. The idea is very simple: Instead of instantiating the dependencies by yourself, keeping all dependencies in a central repository where every unit can ask their dependencies from this repository.

Although it sounds very good, it brings some complexity to your system later in the development phase. One of the major concerns about Dependency Lookup is they are not testable.

Let’s see in the example:

```
// some code..
Reader reader = Repository.load(Reader.class); // this part may
vary
// more code..
```

So as you can see from above code, there is no dependency injection or hard-coded instantiation of the dependency in the code. However, when you try to test this code snippet, you will find yourself in need of using this very same repository. This is exact same problem with static method usages. Any static/global access in the code is always hard to test.

Another problem with Dependency Lookups is you cannot easily see which implementation of this dependency is coming from the repository. In above example, if you ask yourself: “Is this a FileReader, or StringReader or SocketReader?” you will see that it is hard to answer this question when all of these implementations implements the same Reader interface.

Possible solution to this problem is extracting this snippet as a separate method. And when you test this unit, overriding this extracted method and returning a stub/mock will do the magic. Let’s see in an example:

```
// some code..
Reader reader = getSocketReader(); // extract method

protected Reader getSocketReader() {
    return Repository.load(Reader.class);
}

// more code..
```

And when you want to stub this Reader in the test, all you need to do is overriding this method and returning a stub/mock equivalent.

```
class MyFakeUnit extends MyUnit {
    @Override protected Reader getSocketReader() {
        return new MyStubbedSocketReader();
    }
}
```

Now you can use MyFakeUnit in the test instead of MyUnit to isolate this dependency from the test. Ideally you try to avoid using Dependency Lookups. However, if it is there because of some framework usage, above solution will do the trick.

Object Initializer Blocks

Object initialization has two aspects. First one is about the constructor/initialization block, and second one is about the usage of this object. There are certain things that we should avoid while coding the constructor like hidden dependencies, switch cases, if clauses, loops, etc. However, the most major thing that you should be aware of, to me is avoiding Singleton dependency (one instance in entire system), and static method usage.

Developers mostly tend to load some configuration or a socket connection (database, HTTP, remote, etc) in the constructor:

```
class MyUnit {
    private DBConnector db;
    private Configuration config;

    public MyUnit() {
        db = DBConnector.getInstance();
        config = Configuration.Load(Configurations.SYSTEM);
    }
}
```

If you check the above code piece, you will see that db variable is using a Singleton object and config variable is using some static method to initialize itself. So what is wrong with it? Well the problem is isolation of this unit. As you see, it is impossible to inject any fake/mock DBConnector or Configuration class to this unit while we're doing unit testing. If your DBConnector instance is connecting to the production system, then every time you run your unit test, it will try to connect to the production system database which we don't want it for sure.

Another problem with this type of constructors is that it's also causing the "hidden dependency" problem that we described earlier. To solve all these problems, it is better to extract these dependencies and use the extracted methods instead:

```
// some code..
db = getDatabaseConnector();
config = getSystemConfiguration();

// and the extracted parts would be
public DBConnector getDatabaseConnector() {  /* .. */ }
public Configuration getSystemConfiguration() {  /* .. */ }
```

Second problem with object initialization blocks is their usages. It sometimes can be brutal because of the transitive dependencies. Let's see it in an example:

```
// some code..
public CreditCard(PaymentService paymentService, CustomerService
customerService) {
    // details..
}

// and usage of this unit somewhere in the system
PayPalWebService paypalWebService = new PayPalWebService();
PaymentGateway paymentGateway = new PaymentGateway();
paymentGateway.setClearingHouse(paypalWebService);
PaymentService paymentService = new PaymentService();
paymentService.setPaymentGateway(paymentGateway);

ProfileService profileService = new ProfileService();
CustomerService customerService = new
CustomerService(profileService);

CreditCard card = new CreditCard(paymentService,
customerService);
```

As you see, all these transitive dependencies tie you up so bad that you cannot even do a simple initialization. So how do we solve this problem? Well, if it would just be the matter of testing, we would inject fake/mock/stub services to the CreditCard. However, we are talking about the real usage of this CreditCard class. In this case, we have two options. We either eliminate the dependencies, or we will look for a dependency injection framework. My personal choice is focusing on the first one if possible. If you were building an enterprise level system, then going for second option would make more sense. Depending on the language that you are using for development, you can find lots of different frameworks that help you on injecting dependencies.

Unclear Execution Paths

Every time your program sees a conditional clause, it diverts the execution. These diversions are called execution paths. For instance, you would have two separate execution paths for the user who is logged in or not. The problem is not about having different execution paths. It is about abusing the usage of them. Sometimes developers create 4-5 different execution paths from a single point. And on top of this, they might even add some nested execution paths. Check the following example:

```
// some code..
if (lineItem != null && (lineItem.quantity > 0 &&
customer.isActive() && (card.expirityYear > 2010 || card.type ==
AMX)) || couponCode.isValid()) {
    // details..
}
```

Can you tell how many execution paths this particular conditional case creates? I am not even sure if I typed correctly. When this developer leaves the team, this business logic will be very hard to identify. And moreover, how to test, or what to test here? As you see, it is really hard to decide.

“if” usages should be always under control. If we cannot control, it means that we cannot test it. Ideally it is always wise to avoid any of if usages, if it is possible. So I prepared a quick guideline for “if” usages:

Allow them when:

- Condition can be expressed without “AND”, “OR” operators:
  ```
  if (myContainer.hasItems()) {
      //...
  }
  ```

- Condition is about an encapsulated field:
  ```
  if (_visibility == VISIBLE) {
      //...
  }
  ```

- Condition is about non-float numerical comparison:
  ```
  if (items.size() > 2) {
      //...
  }
  ```

Be cautious when:

- There is more than one "AND", "OR" operator:

```
    if (_isVisible && _isParent) {
        //...
    }

    // refactor it:
    if (canAddUser()) {
        //...
    }
```

- There is a negative logic:

```
    if (!_isParent) {
        //...
    }

    // change your logic and make it positive if possible,
    // positive statements are easier to understand
```

- There is a very long condition because of variable/method names:

```
    if
(payPalWebService.isClearingHouseAvailableForPurchasing(myLineIt
em)) {
        //...
    }

    // extract a method and give a short name that tells
    // the purpose of this condition
    if (clearingHouseReady(myLineItem)) {
        //...
    }
```

- There is a "if/else" logic:

```
    if (_isParent) {
        number = generateItemNumber(this);
    } else {
        number = 0;
    }

    // ensure you forced everything to not have this situation.
    // if you still have, try doing something like:
    number = 0;
    if (_isParent) {
        number = generateItemNumber(this);
    }
```

Don't allow when:

- It is just a "NULL" check:

```
    if (user != null) {
        //...
    }

    // try not having a null user in the first place.
    // if you don’t know how to avoid, please refer to my other
article [7]
```

- There are more than two “AND”, “OR” operators:

```
    if ((_isVisible && !isParent) || iAmMaster ||
youCanPassIsSet) {
        //...
    }

    // avoid having multiple conditions in one clause.
    // if you already have, extract a method out of it:
    if (isValid()) {
        //...
    }
```

- There is nested “if/else” statement:

```
    if (isThisTrue) {
        if (checkIfHeIsTheMaster() && !hmmmThisSeemsWeird()) {
            // do something...
        } else {
            // do something else...
        }
    } else {
        // do some other thing...
    }

    // you should never have this situation.
    // try using well known OOP technique called POLYMORPHISM,
    // or if the operator is one of (>, <, ==) then various
possible
    // design patterns like Strategy (behavior modification),
    // Visitor (context evaluation) or Specification (item
selection).
    // if you're not happy, but you don't know what to do then
talk
    // to someone you think he/she might know.
```

If you can follow above guideline, it will be really easy to test your code since identifying execution paths, and creating test coverage for them will be very obvious.

Train Wrecks

“Train Wrecks” is a term used to refer too many object messaging in one line. For instance:

```
object.message(); // this is a normal call
object.someOtherObject.someService.getMeSomeInfo(); // this is a
train wreck
```

As you see, reading the second line and understanding the components of it takes some time. It also violates “Law of Demeter” [8] which has only 3 rules to watch:

- Each unit should have only limited knowledge about other units: only units "closely" related to the current unit.
- Each unit should only talk to its friends; don't talk to strangers.

- Only talk to your immediate friends.

Even though having train wreck expressions seems useful, it could be a real pain when you get an exception in that particular line. Identifying the source of that exception always takes more time than a simple expression. Also your components become tightly coupled. Later in the design, untying these dependencies would be really hard.

So how do we solve this problem? Basically, avoid the train wrecks, and just follow the 3 rules above.

Conclusion

I hope this opens your vision a bit. It is very hard for me to cover everything here. I'm sure all the examples I gave might have exceptional cases depending on the context. However, all these things are all about paying more attention. As you can see from above examples, all of them can be easily done while doing pair programming or code reviews. Writing clean code, making your code testable is not hard; it is very easy indeed if you do pay a little attention to details. And it would make you much better developer. Although being indispensable in the team seems good to many of us, it is actually never good. It just increases your responsibilities, your tasks, and it forces you to work more and more. And it brings you at such a point where no one would want to work with you. So don't be indispensable.

For more information about all these topics, please refer to Refactoring Book [9] or site [10], SOLID Principles, and Clean Code talks [11].

References

1. Growing Object-Oriented Software, Guided by Tests, Steve Freeman and Nat Pryce, Addison-Wesley Professional
2. Clean Code, Robert C. Martin, Prentice Hall
3. Simplest Thing That Could Possibly Work, Ward Cunningham, http://www.artima.com/intv/simplest.html
4. SOLID Principles, Robert C. Martin, http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
5. Extreme Programming Explained, Kent Beck, Addison-Wesley Professional
6. SOLID Principles, Robert C. Martin, http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod
7. How to Avoid Null Pointer Exceptions, Isa Goksu, http://isagoksu.com/2009/development/java/how-to-avoid-nullpointerexceptions-npe/
8. Law of Demeter, Northeastern University, http://en.wikipedia.org/wiki/Law_of_Demeter
9. Refactoring: Improving the Existing Design, Martin Fowler, Addison-Wesley Professional
10. Refactoring, Software Development Community, http://www.refactoring.com
11. Clean Code Talks, Misko Hevery, http://www.youtube.com/view_play_list?p=BDAB2BA83BB6588E

Model-Based Testing Adds Value

Ewald Roodenrijs. ewald.roodenrijs @ sogeti.com
Sogeti Nederland B.V., http://www.sogeti.nl

As a test manager I always hear the same complaints about testers: good testers are hard to come by and they take a lot of time working on their tasks. Both are true. There are not enough good resources available to thoroughly test all the applications that are on the market today and testing can take a lot of time to do it correctly and with the right coverage. There are solutions available to help with these complaints, but they only address one of them: they help you get more good testers or they speed up the testing process. How can we address these two complaints at once?

One of the most time and resource consuming tasks of testing is the creation of test scripts for checking if the software product complies with its specifications. Scripted testing can do this. While testers interpret the specifications, test cases and test scripts are created. Their execution checks if the software was created according to these specifications.

Manual scripting is like using a hammer to wrench a screw into the wall; it can be done more efficiently. Reducing time spend on manual scripting and having less testers involved with execution can create an added value for testing. If this task could be automated it would help those testers who are creating these test cases. Model-Based Testing (MBT) is a method for creating functional test cases.

MBT allows generating test cases from models that describe the test object or system under test (SuT). According to the Dutch test association TestNet (www.testnet.org), Model-Based Testing is:

> "Software testing in which a test model is created that describes some of the expected behavior (usually functional) of the test object. The purpose of creating this test model is to review the requirements thoroughly and/or to derive test cases in whole or in part from the test model. The test models are derived from the requirements or designs."

Some benefits of MBT are:

- MBT provides the opportunity to automate the process for test specification.
- MBT creates acceleration in the specification of test scripts.
- MBT makes the time and resource consuming task of test specification less dependent on the amount and expertise of testers.
- With the use of a model, small changes in the documentation are translated into new test scripts in few seconds.

So MBT seems to be a useful tool for the (future) test market. Testers can use MBT to specify all the functional test cases! MBT can make testing applications easier and faster in the future. But how does MBT actually work and what can you do with it?

Advertisement - ignite Conferences 2010 - Click on ad to reach advertiser web site

iqnite

iqnite Conferences 2010

Together, we bring quality to life.

GERMANY | 2010

27 - 30 April 2010
Dusseldorf, Germany

SUISSE | 2010

15 June 2010
Geneva, Switzerland

SCHWEIZ | 2010

21 September 2010
Zurich, Switzerland

NORDIC | 2010

29 - 30 September 2010
Stockholm, Sweden

UNITED KINGDOM | 2010

4 October 2010
London, United Kingdom

ÖSTERREICH | 2010

13 October 2010
Vienna, Austria

AUSTRALIA | 2010

26 - 27 October 2010
Sydney, Australia

SOUTH AFRICA | 2010

18 November 2010
Johannesburg, South Africa

Methods & Tools
15% Discount
for Subscribers
Promotion Code
MT 2010

iqnite is the leading software quality management and testing conference. A series of annual events worldwide are the meeting place for the testing and quality community where you meet experienced industry insiders and eminent experts sharing their knowledge and presenting the latest industry trends.

www.iqnite-conferences.com

Model, then generate

Model-Based Testing is an approach based on creating test cases derived from a behavior model of the test object: the (test) model. This model describes the expected *behavior* of the test object. Where possible, test cases are then automatically generated from the test object. The challenge with this approach lies in creating a formal behavior model in which the operation of (part of) the application is represented.

Functional requirements, for instance derived from use cases, can serve as a starting point for the creation of the model. The model used for developing the application or a separate test model can be used. A test modeler creates a test model by hand and a tool that will create test cases can read this model. A lot of tools use an UML (Unified Modeling Language) model. UML is a unified accepted language to create models. With UML (2.0) standards, the model is interchangeable with multiple applications that are available on the market. The tools can also improve the process of designing the models.

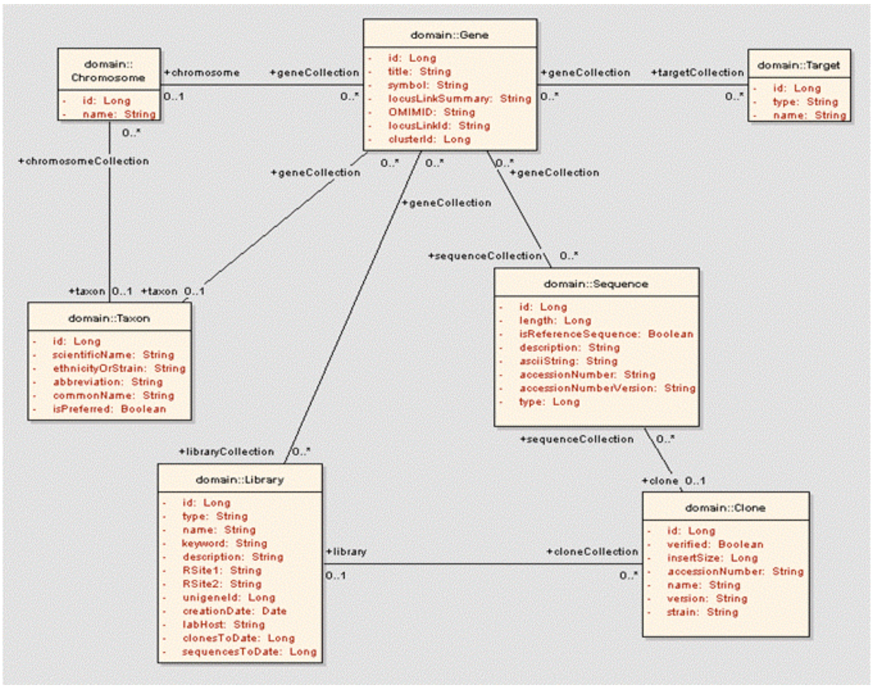


Figure 1 - UML Class Diagram

Using a separate test model has some advantages over taking a development model. The test model is an isolated model, not a model created with a purpose for analysts, designers or developers. The benefits of this approach are:

- The test model contains other topics in the classes than a development model, for instance the use of test data and the use of booleans;
- Errors of development model will not be incorporated in the test model, allowing testers to find defects created during design.

An advantage of using a development model is that no separate model has to be created for testing. Some tools even allow the use of other type of models like pseudo code, decision tables, Markov chains or model specially designed for the tool.

After the creation of a model, a MBT tool can use it. This tool can automatically generate multiple test cases from the model, including traceability to and from different requirements (depending on the MBT tool used). The MBT tool generates test cases by using ‘paths’ through the system on which the test cases are based. These test paths are similar to those used in test design techniques. During the generation of the test cases, some tools can automatically check for consistency. When the consistency of test cases is not achieved, errors should be fixed in the test model.

Because MBT tools generate test cases from a model based on the requirements, it is easy to change test cases when requirements change. After a requirement is changed, you adjust the model and generate the new test cases. This allows a dynamic creation of test cases compared to a more static situation when they are created manually.

Not always automated execution

Model-Based Testing gives testers an excellent option to use the automatically generated test cases for automated test execution. Most tools used for MBT have this ultimately in mind when generating test cases from a model.

Tip: When modeling, appoint some functional ‘keywords’ serving test automation, which can be developed more quickly.

The use of ‘keywords’ in the test model helps the automation of test case execution. Some tools have a plug-in for various test execution tools to manage and execute the generated test cases automatically.

Automated test execution is not always possible or even wanted. So some MBT tools generate test cases into readable steps. These test cases can then be executed manually from the script. In manual execution, I recommend to use keywords for easy reading. Once the keywords are implemented, the test cases can be executed manually. After the creation of the test cases, MBT tools often have an option to export them to multiple formats, such as MS Word or Excel.

Attachment	Step Name	Description	Expected Result
	createCase	Press the Create Case (Label: Insert appointment) button on the menu. with inCanMakeModifications = 'false' with inCanReopenAccount = 'false'	'sut.message' has to be 'NONE'
	Body: createSavingsAccount	Click the Open Savings Account (Label: Open savings account) button.	'sut.message' has to be 'NONE'

Table 1 - Manual test case

Test design techniques covered

When starting a test project, a test strategy is formed based on the wishes of the client, defining result, risk, time and costs. This test strategy is then agreed upon with the client to make sure this is what the he wants. With test design techniques (TDT), the risks of the defined impact of the testing can be implemented in the test execution when validating the test object. Assigning the test design techniques to the test object, the test strategy is executed to validate the test object. Therefore it is important that the test design techniques covered by Model-Based Testing match the generated test cases with the established test strategy.

Using UML models, MBT covers the standardized test design technique Process Cycle Test (PCT). In the PCT tests, increasing or lowering the test depth changes the intensity. It also shows that a MBT tool can cover not only the functional quality characteristic, but also suitability. The same can be said for the Use Case Test (UCT). Because the model can be derived not only from requirements, but also from use cases, it is also easy to cover this test design technique. As with PCT, it also covers suitability.

MBT can cover more than PCT and UCT. UML models used for MBT can be customized to cover other test design techniques. Other types of models can even cover more test design techniques. Using different types of data in the UML model can create various situations, like the ones needed for boundaries and equivalence classes. This creates the possibility to use other test design techniques like Data Combination Test (DCT), Semantic (SEM) and Syntactic Test (SYN), Decision Table Test (DTT) and Elementary Comparison Test (ECT). However, if one of these techniques is chosen to test the software, this should be decoded before creating of the (UML) models.

The logical test cases should be created before creating the model to know how to customize it. With the DCT, SEM and SYN, the use of the correct test data may be sufficient.

Test design technique	**Quality Attribute**	**Standard**	**Possible**
Process Cycle Test	Suitability Functionality	X	
Use Case Test	Suitability	X	
Data Combination Test	Functionality		X
Semantic test	Functionality		X
Syntactic test	Functionality		X
Decision Table Test	Functionality		X
Elementary Comparison Test	Functionality		X

Table 2 - Test design techniques covered with MBT

As an additional advantage, MBT allows the creation of standard test cases. This can be an added value for the ability to outsource the test execution.

Evaluate early

With Model-Based Testing (MBT) it is possible to automatically generated test cases derived from a (test) model. As you derive the models from the requirements, you also evaluate the requirements.

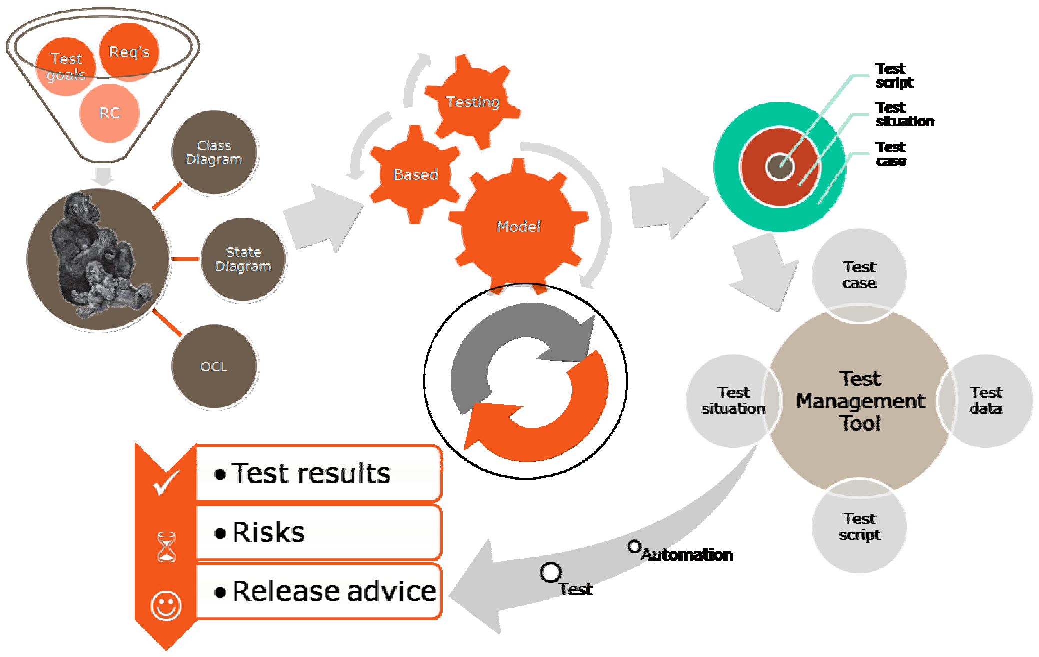
When you create test models, questions will come up about the quality of the functional requirements. These questions are simple discoveries of defects about the requirements. To solve them they need to be presented to the designer or analyst of those requirements. In a normal, non-MBT situation, these issues could only be found during the (manual) specification of the test cases. So these defects could already be added to the code or the prototype. With MBT, these discoveries should already be made after delivering the functional requirements during the preparation phase. It pushes the preparation phase even more upward in the development process.

And now?

Testing doesn’t stop with the automated creation of test cases. The test cases should still be executed. When ‘keywords’ are used to help the test execution, it is also possible to automate this test execution. Using formal UML models test data helps to generate automated test cases. Some tools even have plug-ins for various automated test execution tools. Tools that can be used for MBT are (not extensive):

- Smartesting® Test Designer™
- COVER
- Conformiq Qtronic™
- All4Tec MaTeLo
- Microsoft Spec Explorer

MBT cannot be used to test the whole test object, because it covers only the functional quality attributes and part of suitability. Other quality attributes should be tested otherwise. MBT checks if the functional requirements are covered, but you should use other test methods to cover the whole application.



Using MBT can reduce testing time compared to manual specification of test cases. The challenge lies in learning how to create the models. This is an unexplored territory for most testers and could be a separate specialization within testing. However, if you succeed, automatic test cases generation with automated test case execution will become a realistic possibility.

Sonar

Olivier Gaudin olivier.gaudin @ sonarsource.com,
Freddy Mallet freddy.mallet @ sonarsource.com
SonarSource, http://www.sonarsource.com

What is Sonar ?

Sonar is an open source Platform used by development teams to manage source code quality. Sonar has been developed with a main objective in mind: make code quality management accessible to everyone with minimal effort.

As such, Sonar provides code analyzers, reporting tools, defects hunting modules and TimeMachine as core functionality. But it also embarks a plugin mechanism enabling the community to extend the functionality (more than 35 plugins available), making Sonar the one-stop-shop for source code quality by addressing not only developers but also managers needs.

In terms of languages, Sonar support analysis of Java in the core, but also of Flex (ActionScript 3), PHP, PL/SQL and Cobol through plugins (Open Source or commercial) as the reporting engine is language agnostic.

Since version 2.0, Sonar enables to cover quality on 7 axes and so to report on:

- Duplicated code
- Coding standards
- Unit tests
- Complex code
- Potential bugs
- Comments
- Design and architecture

Sonar can be used for one-off audits, but has been designed to support global continuous improvement strategy on code quality in a company and therefore can be used as a shared central repository for quality management.

Web site: http://sonar.codehaus.org
Product Version: Sonar 2.0
Licence: LGPL V3
Support: http://sonar.codehaus.org/support/
Plugins: http://sonar-plugins.codehaus.org
Commercial plugins: http://www.sonarsource.com/plugins/

Why should you manage source code quality?

A well-written program is a program where the cost of implementing a feature is constant throughout the program's lifetime -- Itay Maman

As a quick intro, this is the best definition of source code quality I could find. It gets even stronger when put the other way around: *a badly written program is a program where the cost of implementing a feature grows throughout time.*

That sounds bad, doesn't it?

We have all seen situations where a new project starts whose objective is to develop from scratch an application in a leading-edge technology. Everything goes very fast; first, second, third release and then all of a sudden, the team's velocity starts to decrease. Fourth release is postponed for the third time, fixing something breaks something else…

What is happening here? Given the symptoms, we can make an assumption that the team is suffering from technical debt amongst other things and that stakeholders are not aware of it and can therefore not deal with it. But this will most probably get resolved as the project is new, has visibility and therefore somebody is going to take care of it (at least we can hope so).

This example was only a starter, as we, IT people, do not work most of our time on applications whose development started less than 6 months ago! Our job is mainly made of upgrades to existing applications. That is where most of the money is spent in IT, where there is less visibility, where often there is a big yearly envelop to do as much as we can, where there are people who are key because they are the only ones able to understand the code, where we have no idea how long a change is going to take, where regressions are frequent and people are scared to make changes. And there is basically no attention whatsoever from the business on that, just do it!

Managing source code quality is all about optimizing ROI as it is going to give you visibility and therefore more control on:

1. how hard maintenance is going to be, what can we expect
2. the fact that things are not getting worse
3. the fact that some attention should be given to critical part of the system, to increase for example coverage by unit tests, suppress cycles, remove duplications

Further more it gives a backup for developers to raise their hand when they believe some refactoring is required that would add a bit to a change but would have good ROI.

How to manage source code quality?

There are seven technical axes that should be looked at when doing source code analysis of a project and Sonar is able to support the management of all seven. In the Sonar team, we like to call them the 7 deadly sins of the developer:

- non respect of coding standards and best practices
- lacking comments in the source code, especially in public APIs
- having duplicated lines of code
- having complex component or/and a bad distribution of complexity amongst components
- having no or low code coverage by unit tests, especially in complex part of the program
- leaving potential bugs
- having a spaghetti design (package cycles…)

The first step when doing source code quality management is really to define which of those axes are important to you and to what extend. Then based on the current situation, a plan should be established to reach the target level (that might be simply to keep a high level of quality). It is very important to start small and go bigger when it gets fully adopted by the whole development team.

Now, let's have a look at how to use Sonar in this approach.

Quality profiles
Sonar enables to manage multiple quality profiles in order to adapt the required level to the type of project (only support, new project, critical application, technical lib…). Managing a profile consists of:

1. activate / deactivate / weight coding rules
2. define thresholds on metrics for automatic alerting
3. define project / profile association

Dashboards
Sonar contains 2 dashboards that give the big picture to get hints where there might be issues and to compare projects:

1. a consolidated view that shows all projects
2. a project dashboard is also available at modules and packages level

Hunting Tools
To confirm that what seems to be an issue is really an issue, Sonar offers a hunting toolset that enables to go from overview to smallest details:

- drill down on every measure displayed to see what is behind
- classes clouds to find less covered classes by unit tests
- hotspots to have on a page the most and the least files
- and a multi-entry (duplication, coverage, violations, tests success…) source viewer to confirm the findings made with the hunting tools

TimeMachine
Knowing where an application stands is very important, but it is even more important to know and understand its evolution. Indeed, what is it worth to know that there is 20% of code coverage by unit tests? Is it good or bad? Is the answer different if two months ago it was 15% or 25%? TimeMachine enables to watch the evolution and replay the past, especially as it records versions of the project

How does Sonar work?

Sonar is made of a fairly simple and flexible architecture that consists of three components:

- A set of source code analyzers that are grouped in a Maven plugin and are triggered on demand. The analyzers use configuration stored in the database. Although Sonar relies on Maven to run analysis, it is capable to analyze Maven and non-Maven projects.
- A database to not only store the results of the analysis, the projects and global configuration but also to keep historical analysis for TimeMachine. 5 database engines are currently supported : Oracle, MySQL, Derby (demo only), PostgreSQL and MS SQLServer

- A web reporting tool to display code quality dashboards on projects, hunt for defects, check TimeMachine and to configure analysis.

As part of its analyzers, Sonar core embarks best of breed tools to find coding rules violations (PMD, Checkstyle), detect potential bugs (Findbugs) and measure coverage by unit tests (Cobertura, Clover). But what makes Sonar truly unique is Squid, its own code analyzer that not only parses source code but also byte code and mixes the results.

Since analysis is run through a Maven plugin, Sonar can be launched easily in "Continuous Integration" environments.

Use case on Apache commons-collection project

A pre-requisite to run Sonar is to have Java and Maven installed on the box. Once this is the case, you can run Sonar in 5 simple steps:

1. Download the distribution from http://sonar.codehaus.org/downloads/ and unzip it

2. Open a console and start the server:

> $SONAR_HOME\bin\windows-x86-32\StartSonar.bat on windows

> $SONAR_HOME/bin/[OS]/sonar.sh on other platforms

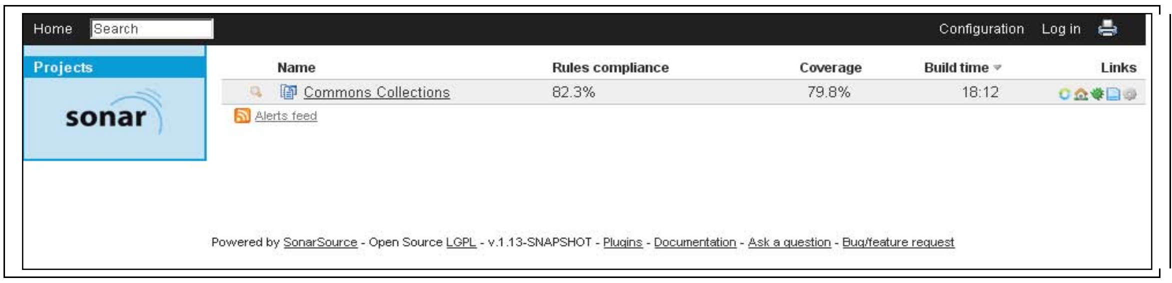
3. Open a console where you want to checkout the source and run:

svn co http://svn.apache.org/viewvc/commons/proper/collections/trunk/ .

4. Run mvn install sonar:sonar in the same directory

5. Browse http://localhost:9000

The home page of the application shows the list of projects under quality control with a few configurable metrics.

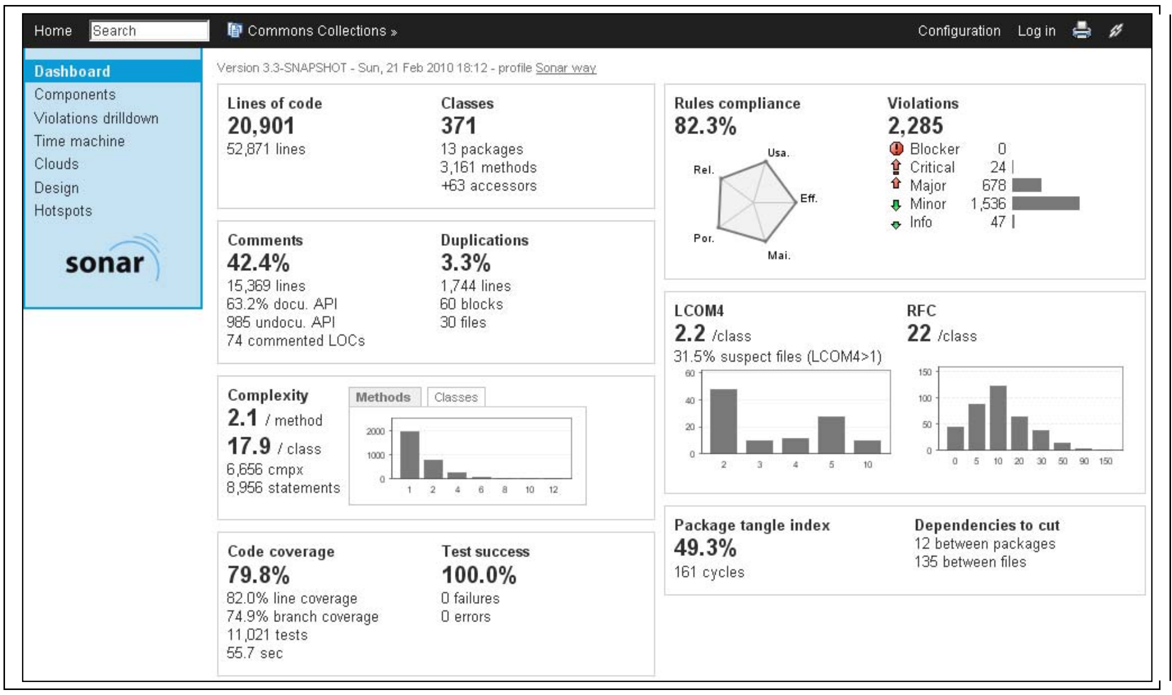


Name	Rules compliance	Coverage	Build time	Links
Commons Collections	82.3%	79.8%	18:12	

Alerts feed

Powered by SonarSource - Open Source LGPL - v.1.13-SNAPSHOT - Plugins - Documentation - Ask a question - Bug/feature request

To zoom in, you can simply click on the project and get its dashboard



Home | Search | Commons Collections » | Configuration | Log in

Dashboard
Components
Violations drilldown
Time machine
Clouds
Design
Hotspots

Version 3.3-SNAPSHOT - Sun, 21 Feb 2010 18:12 - profile Sonar way

Lines of code
20,901
52,871 lines

Classes
371
13 packages
3,161 methods
+63 accessors

Comments
42.4%
15,369 lines
63.2% docu. API
985 undocu. API
74 commented LOCs

Duplications
3.3%
1,744 lines
60 blocks
30 files

Complexity
2.1 / method
17.9 / class
6,656 cmpx
8,956 statements

Code coverage
79.8%
82.0% line coverage
74.9% branch coverage
11,021 tests
55.7 sec

Test success
100.0%
0 failures
0 errors

Rules compliance
82.3%

Violations
2,285

Blocker	0
Critical	24
Major	678
Minor	1,536
Info	47

LCOM4
2.2 /class
31.5% suspect files (LCOM4>1)

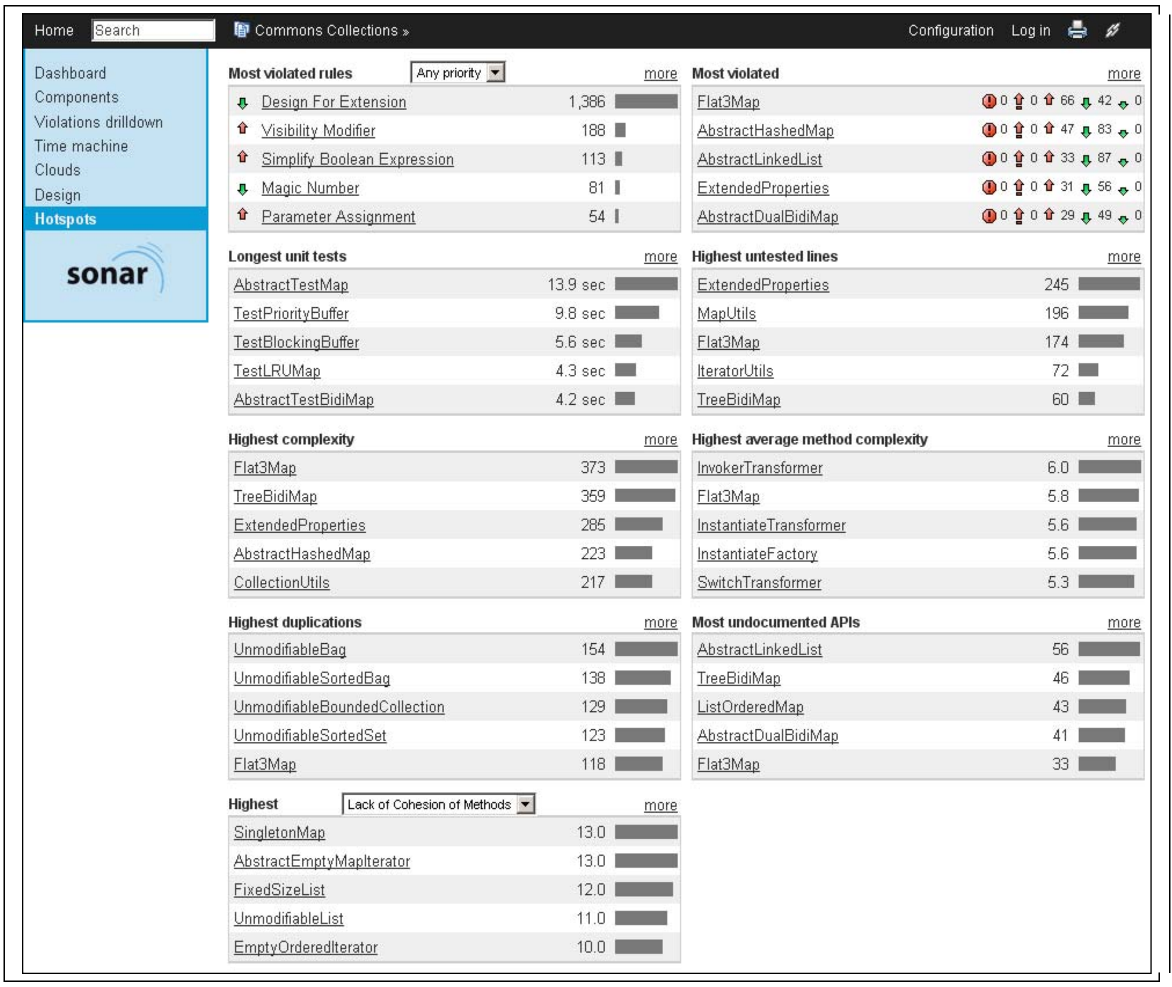
RFC
22 /class

Package tangle index
49.3%
161 cycles

Dependencies to cut
12 between packages
135 between files

From there you have access to a series of hunting tools amongst which:

The hotspot to find out about the files that have "the most" or "the least" …



Home | Search | Commons Collections » | Configuration | Log in

Dashboard
Components
Violations drilldown
Time machine
Clouds
Design
Hotspots

Most violated rules (Any priority) more

Design For Extension	1,386
Visibility Modifier	188
Simplify Boolean Expression	113
Magic Number	81
Parameter Assignment	54

Most violated more

	Blocker	Critical	Major	Minor	Info
Flat3Map	0	0	66	42	0
AbstractHashedMap	0	0	47	83	0
AbstractLinkedList	0	0	33	87	0
ExtendedProperties	0	0	31	56	0
AbstractDualBidiMap	0	0	29	49	0

Longest unit tests more

AbstractTestMap	13.9 sec
TestPriorityBuffer	9.8 sec
TestBlockingBuffer	5.6 sec
TestLRUMap	4.3 sec
AbstractTestBidiMap	4.2 sec

Highest untested lines more

ExtendedProperties	245
MapUtils	196
Flat3Map	174
IteratorUtils	72
TreeBidiMap	60

Highest complexity more

Flat3Map	373
TreeBidiMap	359
ExtendedProperties	285
AbstractHashedMap	223
CollectionUtils	217

Highest average method complexity more

InvokerTransformer	6.0
Flat3Map	5.8
InstantiateTransformer	5.6
InstantiateFactory	5.6
SwitchTransformer	5.3

Highest duplications more

UnmodifiableBag	154
UnmodifiableSortedBag	138
UnmodifiableBoundedCollection	129
UnmodifiableSortedSet	123
Flat3Map	118

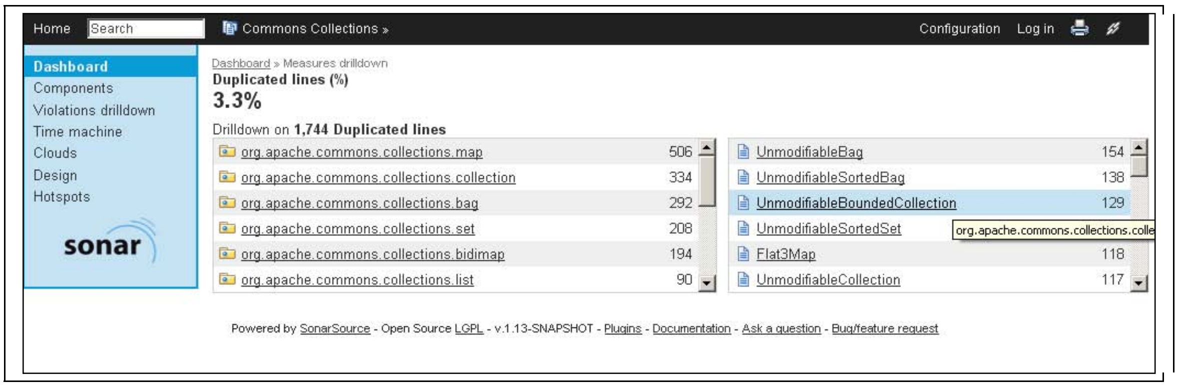
Most undocumented APIs more

AbstractLinkedList	56
TreeBidiMap	46
ListOrderedMap	43
AbstractDualBidiMap	41
Flat3Map	33

Highest (Lack of Cohesion of Methods) more

SingletonMap	13.0
AbstractEmptyMapIterator	13.0
FixedSizeList	12.0
UnmodifiableList	11.0
EmptyOrderedIterator	10.0

But also, any metric in the dashboard is clickable to jump to behind the scene and get a view of the metric by underlying component



Dashboard » Measures drilldown
Duplicated lines (%)
3.3%

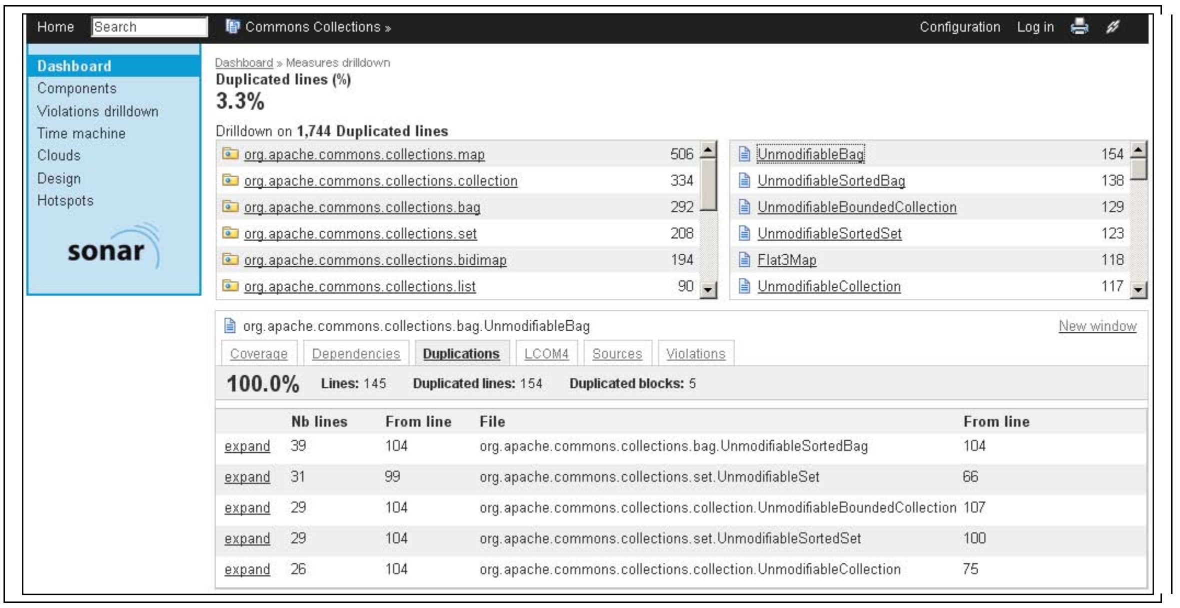
Drilldown on 1,744 Duplicated lines

org.apache.commons.collections.map	506
org.apache.commons.collections.collection	334
org.apache.commons.collections.bag	292
org.apache.commons.collections.set	208
org.apache.commons.collections.bidimap	194
org.apache.commons.collections.list	90

UnmodifiableBag	154
UnmodifiableSortedBag	138
UnmodifiableBoundedCollection	129
UnmodifiableSortedSet	org.apache.commons.collections.colle
Flat3Map	118
UnmodifiableCollection	117

Powered by SonarSource - Open Source LGPL - v.1.13-SNAPSHOT - Plugins - Documentation - Ask a question - Bug/feature request

Each hunting tool eventually brings the hunter to the source code where the preys are highlighted



Dashboard » Measures drilldown
Duplicated lines (%)
3.3%

Drilldown on 1,744 Duplicated lines

org.apache.commons.collections.map	506
org.apache.commons.collections.collection	334
org.apache.commons.collections.bag	292
org.apache.commons.collections.set	208
org.apache.commons.collections.bidimap	194
org.apache.commons.collections.list	90

UnmodifiableBag	154
UnmodifiableSortedBag	138
UnmodifiableBoundedCollection	129
UnmodifiableSortedSet	123
Flat3Map	118
UnmodifiableCollection	117

org.apache.commons.collections.bag.UnmodifiableBag   New window

Coverage | Dependencies | Duplications | LCOM4 | Sources | Violations

100.0% Lines: 145 Duplicated lines: 154 Duplicated blocks: 5

	Nb lines	From line	File	From line
expand	39	104	org.apache.commons.collections.bag.UnmodifiableSortedBag	104
expand	31	99	org.apache.commons.collections.set.UnmodifiableSet	66
expand	29	104	org.apache.commons.collections.collection.UnmodifiableBoundedCollection	107
expand	29	104	org.apache.commons.collections.set.UnmodifiableSortedSet	100
expand	26	104	org.apache.commons.collections.collection.UnmodifiableCollection	75

The Sonar Ecosystem

There is a very dynamic ecosystem around Sonar

- An active community made of 300+ people on the user mailing list and 150+ people on the development mailing list
- 35+ plugins on the forge (http://docs.codehaus.org/display/SONAR/Sonar+Plugin+Library/) that are divided into four categories
- Integration with external tools such as Jira, Hudson, Bamboo, GateIn, AnthillPro, Crowd
- Direct extension of core functionality by adding new behavior, calculate advanced metrics or consolidate projects, add new metrics
- Add coverage of languages such as PL/SQL, ActionScript3
- Integration with IDEs to get defects information on the code when it is edited
- 3,000+ downloads per month
- A core development team led by SonarSource (http://www.sonarsource.com)

Conclusion

After the massive adoption of continuous integration engines and Test Driven Development practices, managing source quality looks like the natural next step for development teams in their effort of industrialization. Sonar enables to reach this objective with few efforts and with fun.

In 2010, the Sonar platform is going to continue to evolve, the main axes of development being covering new languages and improving integration with IDEs.

To stay connected, you can follow the Sonar blog : http://sonar.codehaus.org/category/blog/.

Express - Agile Project Management

Adam Boas, http://www.oneadam.net/

Express is an agile project management tool focused on iteration and backlog management. Express is designed around the Scrum approach to Agile Project Management and supports stories with tasks. It is a highly usable application that allows users to quickly become very productive for product backlog management and sprint planning. Developers and other stakeholders are able to track the progress of each iteration using the iteration burndown chart and a virtual wall which shows each story's tasks moving through the various stages from open, through in-progress and test, to done.

The server component of Express is a Spring based Java application which provides all the scalability and reliability of applications built on this robust platform. For those technically minded, the underlying persistence technology is Hibernate's implementation of JPA. The client component is a Flex 3 Rich Internet Application (RIA). This provides an immersive and intuitive user environment quite unlike many HTML based solutions to this problem.

Web site: http://agileexpress.sourceforge.net
Version Tested: 0.7.5 deployed to Tomcat 6.0.20 with Postgres 8.4 on OS X
License & Pricing: Open Source (Mozilla Public License 1.1)
Support: Sourceforge trackers and mailing list

Installation

The Express server component will run on any server that runs Java 5 or above. Typical deployment is to Tomcat 6.x and the developers test on Tomcat 6.0.20 and Jetty. The application comes as a Java web archive (war) file. The current version is called express-0.7.5.war and because of the way the flex client application is compiled, it needs to be deployed at the web root of the container. For Tomcat, this means renaming the war to ROOT.war and putting it in the webapps directory.

Express requires a relational database. It comes with drivers and configuration settings for Postgres and MySQL. It is configured for Postgres by default out of the box (see configuration details below for how to change this). It also requires access to a mail server for sending notifications such as registration confirmation and project access requests.

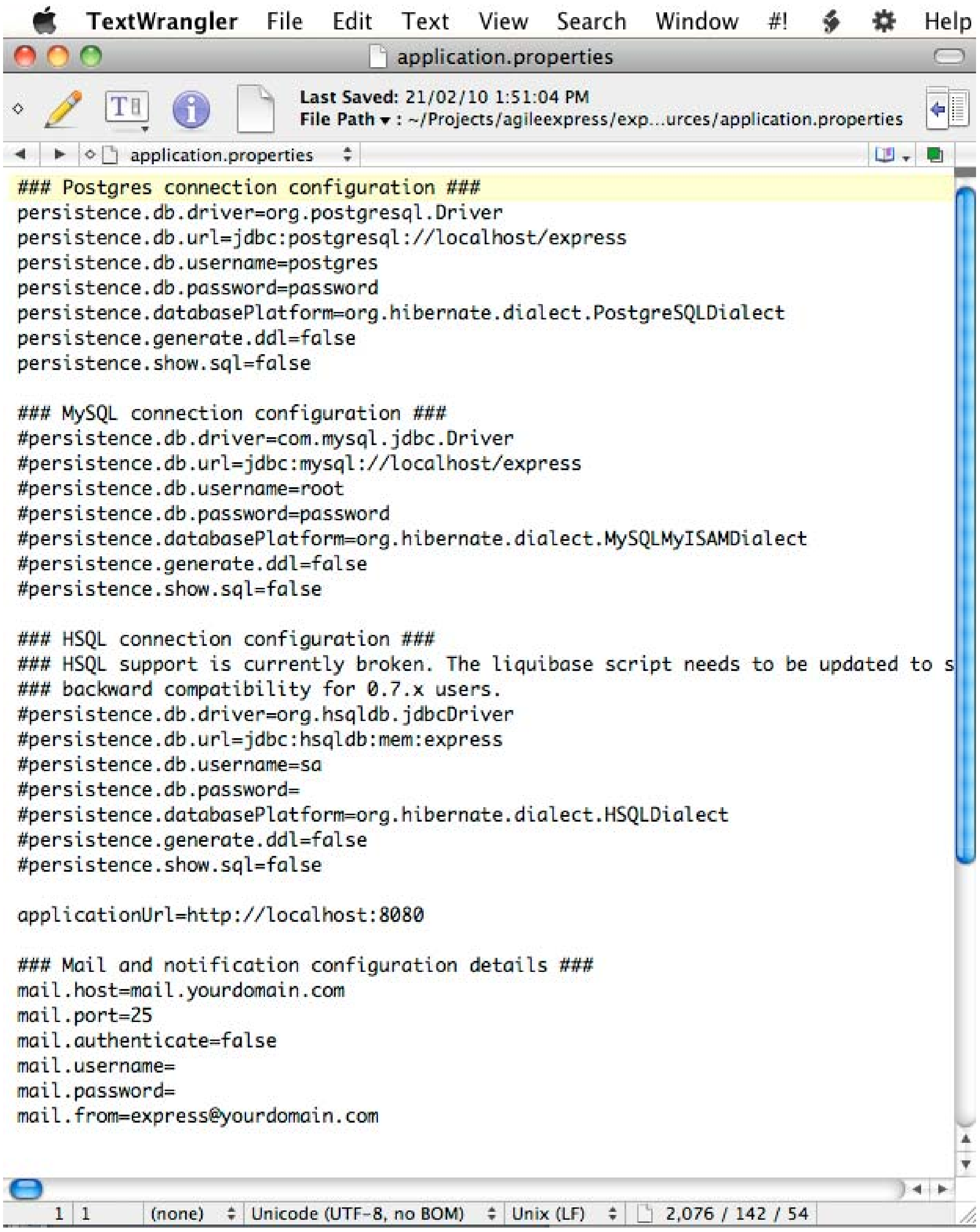
Documentation

Documentation is not extensive. Basic documentation exists on the project website and is available as web pages in the installed application. The application is very focused on backlog and iteration management, and for people familiar with Agile and Scrum, using Express is fairly self-explanatory.

Configuration

Most configuration settings are in the application.properties file, which is located in the classes directory under the WEB-INF folder in the exploded war. There are entries for both Postgres and MySQL databases, by default the MySQL entries are commented out with a '#' at the beginning of the line which enables Express to use Postgres. To use MySQL you comment out the Postgres lines and remove the comment marks from the MySQL entries. In most cases, assuming the default database name of *express* is used, the only items that need to be changed are the hostname for the database (*localhost*) by default, and the username and password to access the database server.

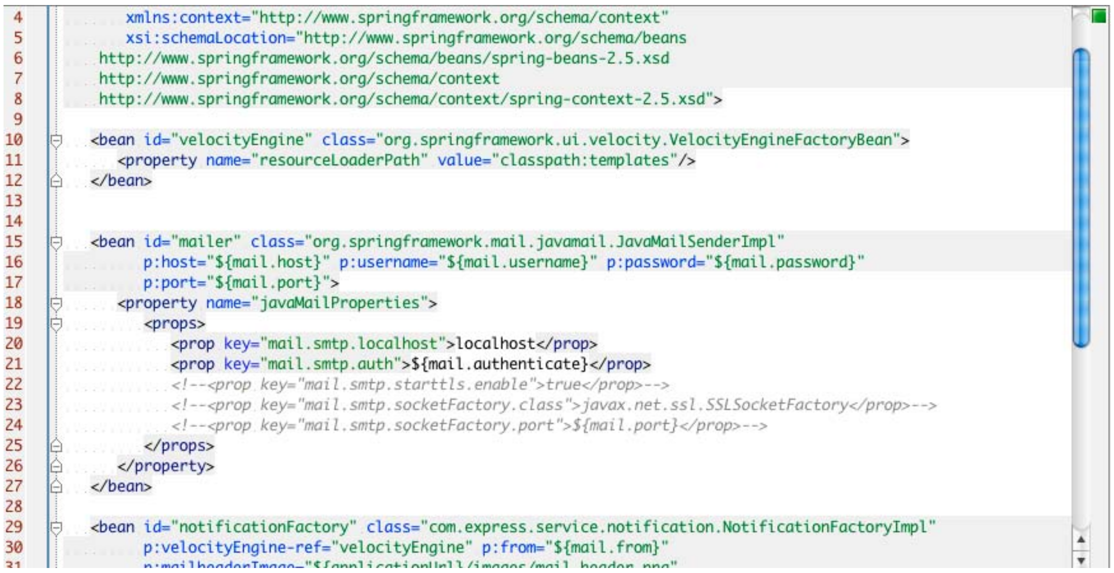
The database name in the screenshot below is *express. Y*ou will need to create a database called express and make sure that the user you define has permission to create, remove, and alter tables in that database. When Express starts up the first time it will create all tables and sequences required.


TextWrangler File Edit Text View Search Window #! Help
application.properties
Last Saved: 21/02/10 1:51:04 PM
File Path ▾ : ~/Projects/agileexpress/exp...urces/application.properties
application.properties
1 1 (none) Unicode (UTF-8, no BOM) Unix (LF) 2,076 / 142 / 54

```
### Postgres connection configuration ###
persistence.db.driver=org.postgresql.Driver
persistence.db.url=jdbc:postgresql://localhost/express
persistence.db.username=postgres
persistence.db.password=password
persistence.databasePlatform=org.hibernate.dialect.PostgreSQLDialect
persistence.generate.ddl=false
persistence.show.sql=false

### MySQL connection configuration ###
#persistence.db.driver=com.mysql.jdbc.Driver
#persistence.db.url=jdbc:mysql://localhost/express
#persistence.db.username=root
#persistence.db.password=password
#persistence.databasePlatform=org.hibernate.dialect.MySQLMyISAMDialect
#persistence.generate.ddl=false
#persistence.show.sql=false

### HSQL connection configuration ###
### HSQL support is currently broken. The liquibase script needs to be updated to s
### backward compatibility for 0.7.x users.
#persistence.db.driver=org.hsqldb.jdbcDriver
#persistence.db.url=jdbc:hsqldb:mem:express
#persistence.db.username=sa
#persistence.db.password=
#persistence.databasePlatform=org.hibernate.dialect.HSQLDialect
#persistence.generate.ddl=false
#persistence.show.sql=false

applicationUrl=http://localhost:8080

### Mail and notification configuration details ###
mail.host=mail.yourdomain.com
mail.port=25
mail.authenticate=false
mail.username=
mail.password=
mail.from=express@yourdomain.com
```

The applicationUrl property (shown above with the value http://localhost:8080) needs to point to the URL of where you have installed express. This URL is used in email notifications sent by the system. It is important so that registration confirmations can be sent and validated to give users access.

Mail settings are also required for notifications to be sent correctly. Username and password fields in mail will be ignored unless mail.authenticate is set to true. For mail providers like Google, who require an SSL connection, some further editing needs to be done in the applicationContextNotification.xml file shown below.



```xml
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.5.xsd">

    <bean id="velocityEngine" class="org.springframework.ui.velocity.VelocityEngineFactoryBean">
        <property name="resourceLoaderPath" value="classpath:templates"/>
    </bean>


    <bean id="mailer" class="org.springframework.mail.javamail.JavaMailSenderImpl"
            p:host="${mail.host}" p:username="${mail.username}" p:password="${mail.password}"
            p:port="${mail.port}">
        <property name="javaMailProperties">
            <props>
                <prop key="mail.smtp.localhost">localhost</prop>
                <prop key="mail.smtp.auth">${mail.authenticate}</prop>
                <!--<prop key="mail.smtp.starttls.enable">true</prop>-->
                <!--<prop key="mail.smtp.socketFactory.class">javax.net.ssl.SSLSocketFactory</prop>-->
                <!--<prop key="mail.smtp.socketFactory.port">${mail.port}</prop>-->
            </props>
        </property>
    </bean>

    <bean id="notificationFactory" class="com.express.service.notification.NotificationFactoryImpl"
            p:velocityEngine-ref="velocityEngine" p:from="${mail.from}"
```

Uncomment the 3 lines commented out above. For those unfamiliar with XML commenting, uncomment means removing the !—characters directly after the < at the beginning of the line and the -- characters directly before the > at the end of the line.

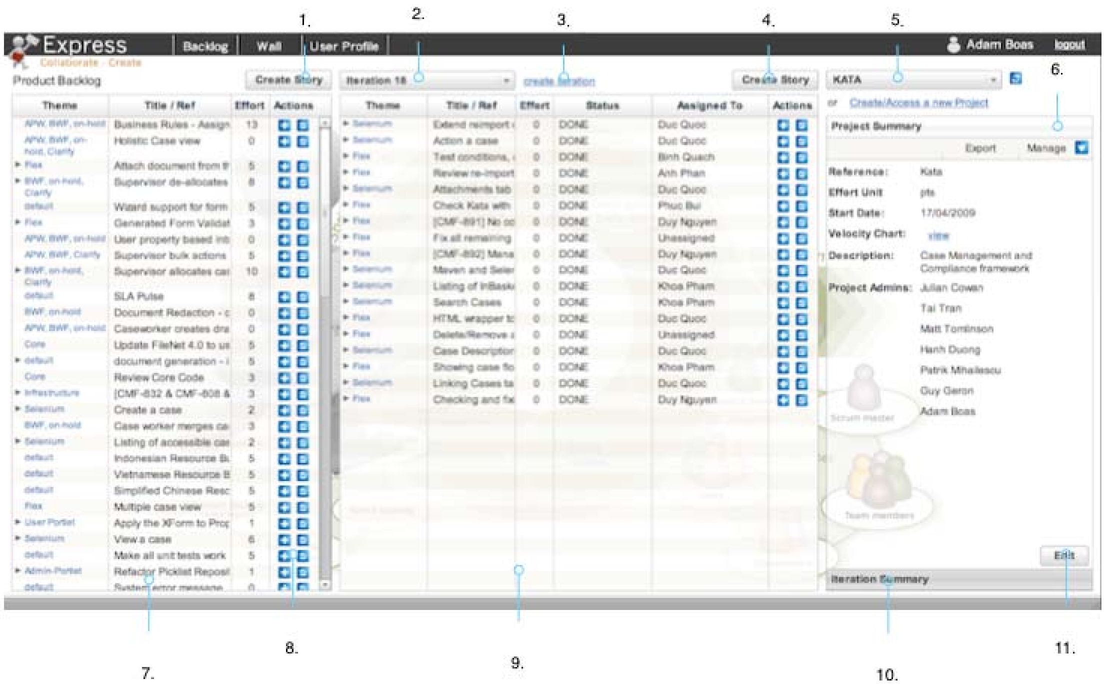
Usage

Express has 2 main screens for daily use: Backlog management, and Virtual Wall. Additionally there are screens for Project creation and managing user details, but a typical user will rarely visit these.

The Backlog management screen is designed for creating the product backlog for a project and allocating the stories from there into iterations, where they will be worked on. Stories can be simply allocated to an iteration, either singly or in groups, by dragging them over into the iteration backlog. Tasks can also be added to stories here and both stories and tasks can be edited to groom the backlog and fill in missing information such as effort estimation, prioritization and grouping in themes.

Beside the project backlog and iteration backlog lists is an accordion panel that has project summary information and iteration summary information. In the project summary panel, besides a useful summary of the project information and access to the velocity comparison chart, users can find controls to export the product backlog to CSV, manage User access and permissions on the project as well as managing project themes. Themes allow the grouping of stories and tasks arbitrarily. Common usages are to group stories into releases or into functional areas.

The Iteration summary panel contains information about the iteration and metrics associated with it as well as allowing the selected iteration backlog to be exported to CSV or printed to cards. Printing is divided into printing Stories and Tasks so that teams who use different coloured cards for stories and tasks can change paper colour.



Product Backlog

Theme	Title / Ref	Effort	Actions
APW, BWF, on-hold	Business Rules - Assign	13	
APW, BWF, on-hold, Clarify	Holistic Case view	0	
Flex	Attach document from th	5	
BWF, on-hold, Clarify	Supervisor de-allocates	8	
default	Wizard support for form	5	
Flex	Generated Form Validat	3	
APW, BWF, on-hold	User property based int	0	
APW, BWF, Clarify	Supervisor bulk actions	5	
BWF, on-hold, Clarify	Supervisor allocates ca	10	
default	SLA Pulse	8	
BWF, on-hold	Document Redaction - c	0	
APW, BWF, on-hold	Caseworker creates dra	0	
Core	Update FileNet 4.0 to us	5	
default	document generation - i	5	
Core	Review Core Code	3	
Infrastructure	[CMF-832 & CMF-808 &	3	
Selenium	Create a case	2	
BWF, on-hold	Case worker merges ca	3	
Selenium	Listing of accessible cas	2	
default	Indonesian Resource Bu	5	
default	Vietnamese Resource B	5	
default	Simplified Chinese Reso	5	
Flex	Multiple case view	5	
User Portlet	Apply the XForm to Pro	1	
Selenium	View a case	6	
default	Make all unit tests work	5	
Admin-Portlet	Refactor Picklist Reposi	1	
default	System error message	0	

Theme	Title / Ref	Effort	Status	Assigned To	Actions
Selenium	Extend reimport	0	DONE	Duc Quoc	
Selenium	Action a case	0	DONE	Duc Quoc	
Flex	Test conditions,	0	DONE	Binh Quach	
Flex	Review re-import	0	DONE	Anh Phan	
Selenium	Attachments tab	0	DONE	Duc Quoc	
Flex	Check Kata with	0	DONE	Phuc Bui	
Flex	[CMF-891] No co	0	DONE	Duy Nguyen	
Flex	Fix all remaining	0	DONE	Unassigned	
Flex	[CMF-892] Mana	0	DONE	Duy Nguyen	
Selenium	Maven and Seler	0	DONE	Duc Quoc	
Selenium	Listing of InBask	0	DONE	Khoa Pham	
Selenium	Search Cases	0	DONE	Khoa Pham	
Flex	HTML wrapper fo	0	DONE	Duc Quoc	
Flex	Delete/Remove	0	DONE	Unassigned	
Selenium	Case Description	0	DONE	Duc Quoc	
Flex	Showing case flo	0	DONE	Khoa Pham	
Selenium	Linking Cases ta	0	DONE	Duc Quoc	
Flex	Checking and fix	0	DONE	Duy Nguyen	

or Create/Access a new Project

Project Summary

Export Manage

Reference:	Kata
Effort Unit	pts
Start Date:	17/04/2009
Velocity Chart:	view
Description:	Case Management and Compliance framework
Project Admins:	Julian Cowan
	Tai Tran
	Matt Tomlinson
	Hanh Duong
	Patrik Mihailescu
	Guy Geron
	Adam Boas

Edit

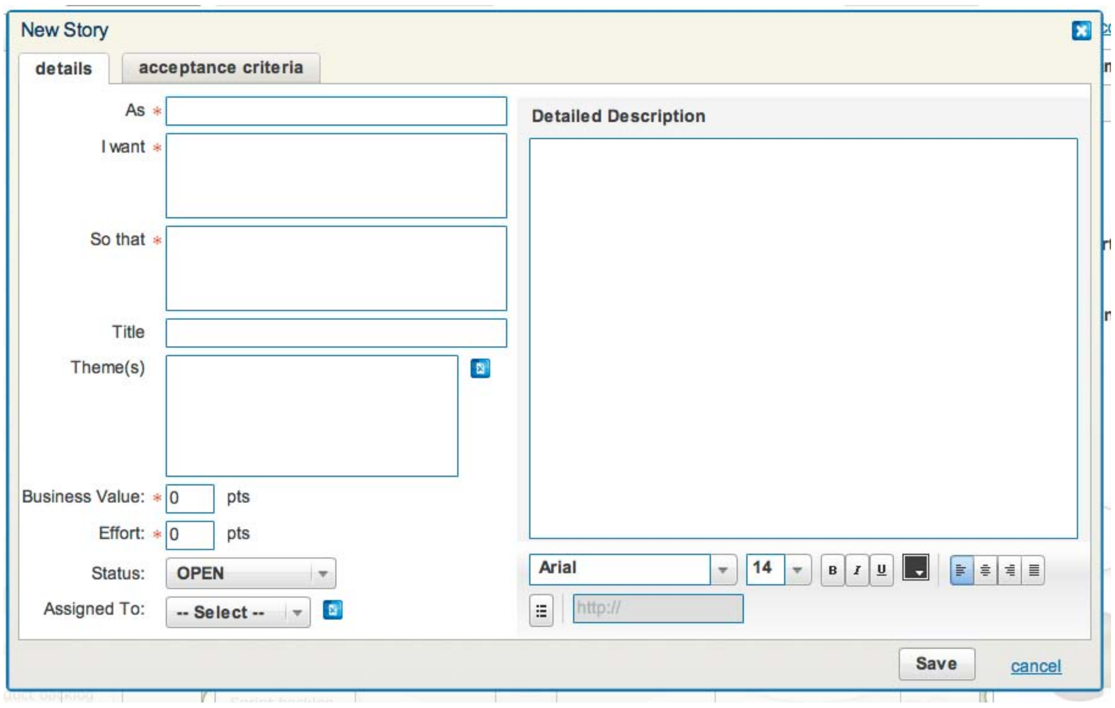
Iteration Summary

Notes

1. Creates a story in the product backlog (data entry via a modal window).
2. Iteration selection drop down, shows all iterations for the selected project
3. Option to create a new iteration for the selected project (only available if the user is an iteration admin for the selected project)
4. Creates a story in the currently selected iteration backlog
5. Project selection drop down
6. Project summary – shows information about the project, allows export of the product backlog to csv and management of developers and themes for the project
7. Product backlog list. Items can be dragged from the product backlog into the selected iteration and vice versa
8. Action buttons allow a task to be added to a story or for a story to be deleted
9. Iteration backlog list
10. Iteration summary when selected the accordion slides up to display the summary data for the iteration including the iteration burndown chart and allows export of the iteration stories to csv
11. Edit project button

Whenever a new story is added or an existing one double clicked the story editing modal window appears. It allows detailed information about the story to be added and, for stories, enforces that the story be described in the usual Agile way: As…I want…so that…

The story editing screen has a second tab for entering acceptance criteria. Each acceptance criteria has a verified field which can be checked by testers when the acceptance criteria has been tested.


New Story

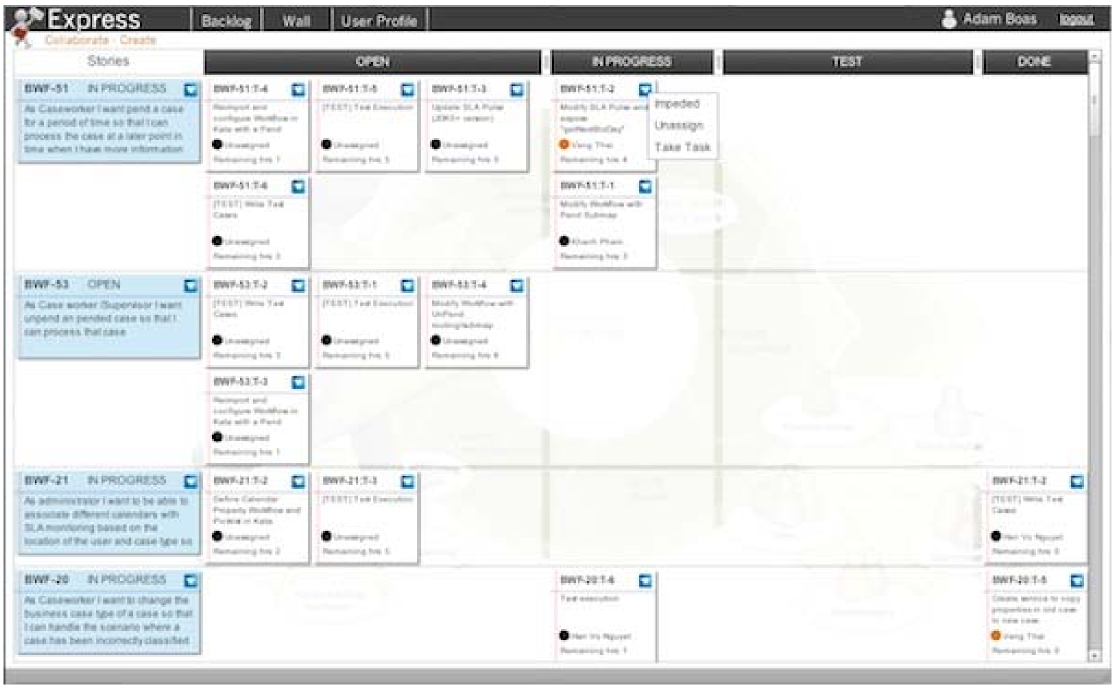
details | acceptance criteria

As *	
I want *	
So that *	
Title	
Theme(s)	
Business Value: *	0 pts
Effort: *	0 pts
Status:	OPEN
Assigned To:	-- Select --

Detailed Description

The virtual wall is the typical team view. On this screen the stories are displayed on virtual index cards, their tasks are laid out in front of them in their swim lanes (OPEN, IN PROGRESS, TEST, and DONE). Developers then take tasks and move them through the swim lanes until they are complete. Stories and tasks can be edited from this screen as well as from the Backlog management screen.

Story and Task cards have quick menus on them to provide short cuts to commonly used functions such as marking the task as blocked or impeded, taking the task or unassigning it. Stories and tasks displayed on the wall are editable by double clicking them in the same way as they are in the backlog management view. Swim lanes on the wall are resizable to allow busy swim lanes to accommodate more tasks horizontally and conserve vertical space so more stories can be viewed concurrently on the screen.



The project creation screen allows new projects to be created or for a user to request access to an existing project. New projects are automatically created and project access requests are emailed to the project admins of the project selected. The admin can then either confirm or reject the user’s request.

Conclusion

Express focuses on doing a small set of functions well, and provides a set of extra features around these functions to help speed up these core tasks. The drag and drop functionality in the backlog management area makes assigning to iterations quick and easy. The reports it provides, such as the burndown chart and velocity comparison chart, are very useful and there are more reports scheduled in the product roadmap.

For teams using Scrum, particularly when there are people not constantly co-located, Express is a very useful tool. The virtual wall allows developers and stakeholders to quickly see how the iteration is progressing. The user interface is snappy and responsive and small features like requesting the person a task is assigned to when that task is dragged into the “IN PROGRESS” lane when unassigned really helps to get things done quickly so that developers can get back to developing.

Express is well worth a look for any team developing using Agile. At present it’s use is restricted to those able to install a Java web application but there are plans in the future to release a version bundled with Tomcat to make installation an easier process and lower the barrier to entry.

Apache JMeter

Sander Stevens, TechTest, http://www.techtest.nl/

Apache JMeter is open source software, a 100% pure Java desktop application designed to (load) test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

Web Site: http://jakarta.apache.org/jmeter/index.html
Version Tested: jmeter 2.3.4 tested from October 2009 until February 2010
License & Pricing: Open Source
Support: User mailing list (jmeter-user@jakarta.apache.org)

What can I do with it?

Apache JMeter may be used to test functional and performance both on static and dynamic resources (files, Servlets, Perl scripts, Java Objects, Data Bases and Queries, FTP Servers and more). It can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types. You can also use it perform a functional test on websites, databases, LDAPs, webservices etc.

JMeter is not a browser. As far as web-services and remote services are concerned, JMeter looks like a browser (or rather, multiple browsers); however JMeter does not perform all the actions supported by browsers. In particular, JMeter does not execute the Javascript found in HTML pages. Nor does it render the HTML pages as a browser does (it's possible to view the response as HTML etc, but the timings are not included in any samples, and only one sample in one thread is ever viewed at a time).

Installation

The installation of JMeter (if you would call it an installation) is pretty straightforward. On the website is a link to the download area of stable versions. You also have the possibility to use nightly builds, but this is at your own risk. No guarantee that they work properly. So the advice is always to start with a stable version.

Download the latest version (zip or tgz) and unpack the archive to a local folder. Before starting JMeter it is wise to have a look at the configuration. The jmeter.properties file (located in the bin folder) contains a lot of settings. Most of those settings should be fine for the average user. The file is well documented and easy to read (and change where needed).

Also pay attention to needed additional jars. If you are going to use JDBC, JMS or JavaMail, the additional jars are not included in the JMeter installation. The user guide on the website explains how to use it and where to get it.

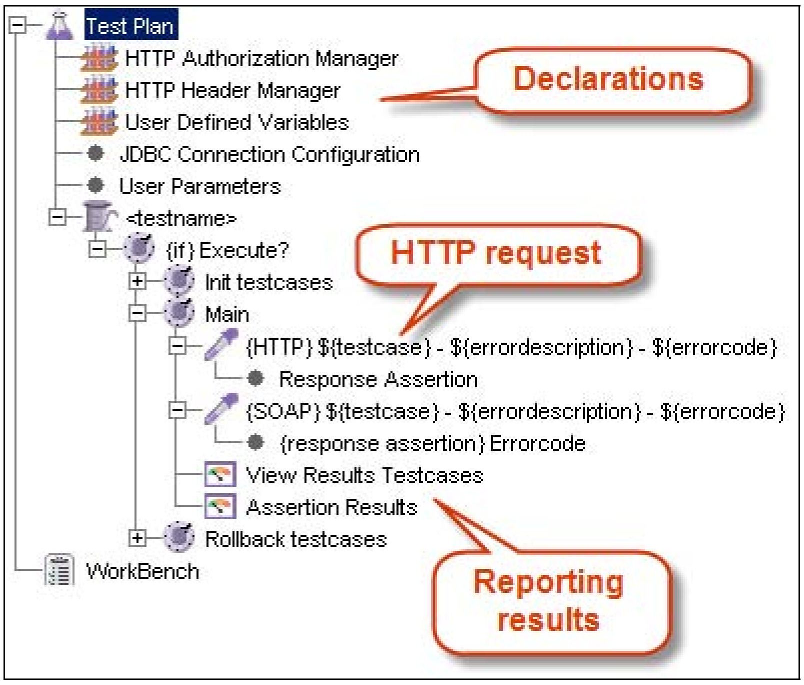
Starting JMeter is just a matter of double clicking jmeter.bat in the bin folder.

Documentation

There is an extensive documentation on the web site in all kinds of ways: user manual, wikis, docs and user experiences. The user manual describes how to create specific testplans (Web, Database, JMS, Webservice, etc) including step by step instructions, examples, bitmaps and tips. Also additional information is described in the user manual like best practices, information about regular expressions and how to use the variables and predefined functions. And last but not least: the component reference describes in detail how every component can be used.

The principle of JMeter

The principle of JMeter is very simple. If you want to test e.g. a SOAP interface layer, all you basically need is the URL and SOAP request. Starting with that you can build your test plan. And this can be as fancy as you want. Using variables, counters, parameters, CSV files, loops, logs, etc. There are almost no limits in designing your test and making it as maintainable as possible.



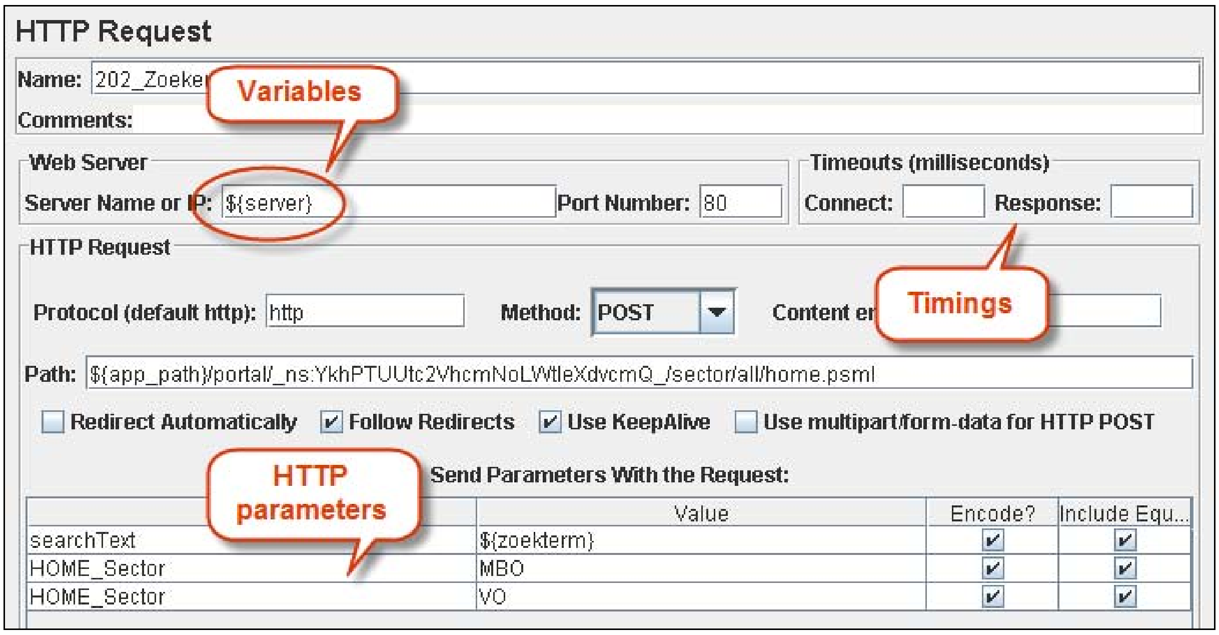
```
Test Plan
    HTTP Authorization Manager
    HTTP Header Manager
    User Defined Variables
    JDBC Connection Configuration
    User Parameters
    <testname>
        {if} Execute?
            Init testcases
            Main
                {HTTP} ${testcase} - ${errordescription} - ${errorcode}
                    Response Assertion
                {SOAP} ${testcase} - ${errordescription} - ${errorcode}
                    {response assertion} Errorcode
                View Results Testcases
                Assertion Results
            Rollback testcases
WorkBench
```

Figure 1 - Snapshot of the JMeter interface

Let’s assume you want to test a web server and fire off some http requests. Figure 1 shows a sample testplan. The testplan starts with some declarations for authorization and user defined variables. Also the JDBC connection is included. If a database is involved, you can set the database configuration and username/password. Don’t forget to included the correct jars!

The HTTP request has some variables in its name. The name of the HTTP request will be displayed in the Result tree and therefore it is handy to include some kind of reference to the test you are running. In this example the testcase number, error description and error code is used to uniquely identify the request.

Figure 2 shows the detail pane of the HTTP request. You can add all the needed information like server name, portnumber, protocol (HTTPS is supported!) and parameters.



HTTP Request

Name: 202_Zoeker

Comments:

Web Server

Server Name or IP: ${server} Port Number: 80

Timeouts (milliseconds)

Connect: Response:

HTTP Request

Protocol (default http): http Method: POST Content en

Path: ${app_path}/portal/_ns:YkhPTUUtc2VhcmNoLWtleXdvcmQ_/sector/all/home.psml

☐ Redirect Automatically ☑ Follow Redirects ☑ Use KeepAlive ☐ Use multipart/form-data for HTTP POST

Send Parameters With the Request:

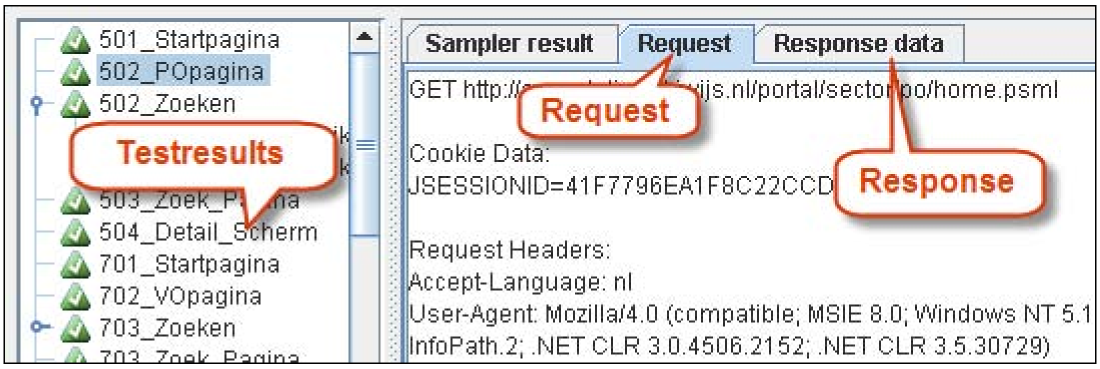
	Value	Encode?	Include Equ...
searchText	${zoekterm}	✓	✓
HOME_Sector	MBO	✓	✓
HOME_Sector	VO	✓	✓

Figure 2 - Adding an HTTP request to the testplan

With regular expressions you can extract for example session ids, store them in a variable and use them in following requests.

Reporting

For reporting purposes JMeter offers several possibilities. The most common is the View Results Tree. This tree will display the requests and has the ability to show the request and the response. The response can be displayed as text, XML, HTML or JSON. The component View Results Tree is mostly used for functional testing.



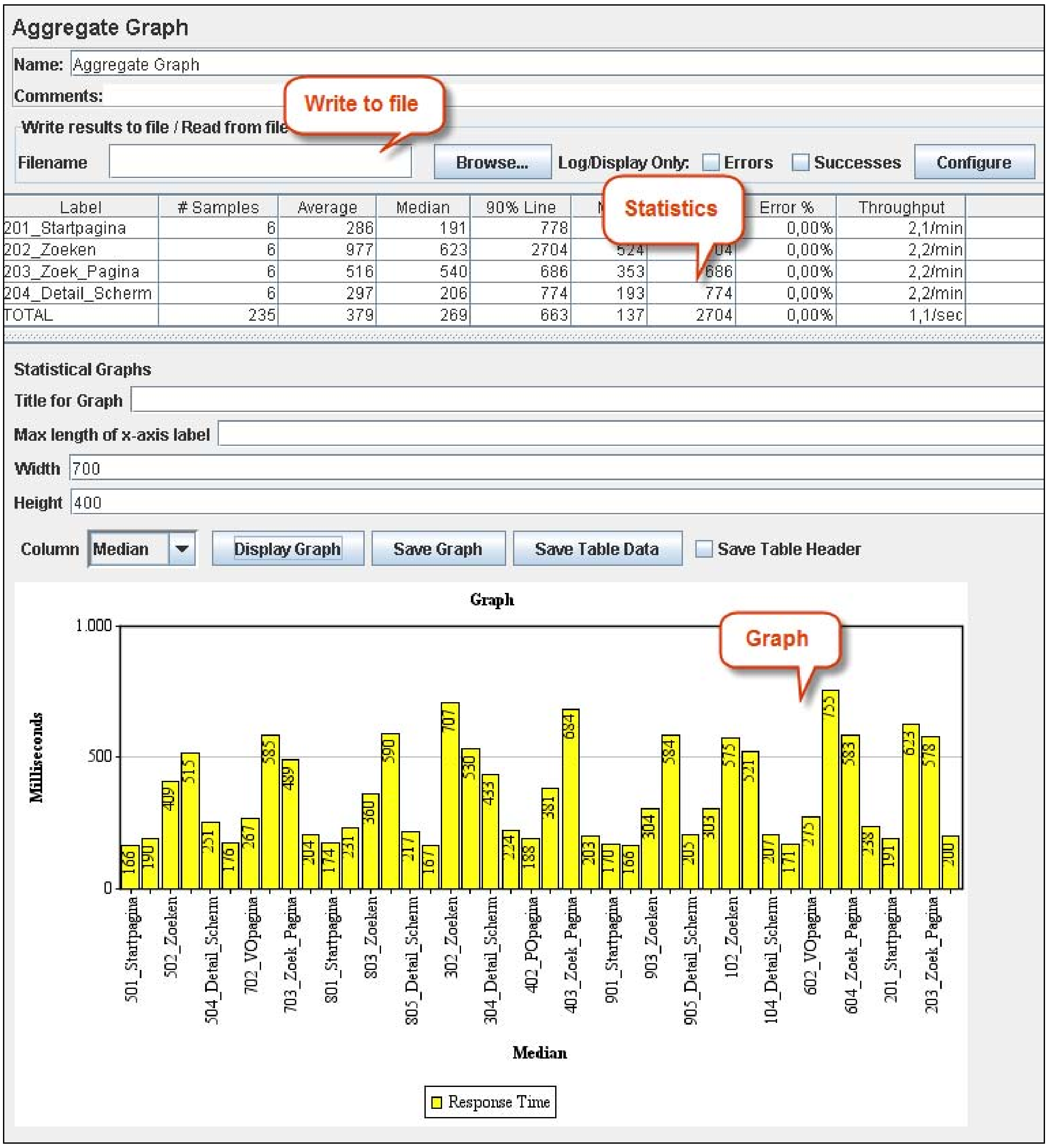
501_Startpagina
502_POpagina
502_Zoeken
503_Zoek_Pagina
504_Detail_Scherm
701_Startpagina
702_VOpagina
703_Zoeken
703_Zoek_Pagina

```
GET http://                    ijs.nl/portal/sector/po/home.psml

Cookie Data:
JSESSIONID=41F7796EA1F8C22CCD

Request Headers:
Accept-Language: nl
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1
InfoPath.2; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
```

Figure 3 - A snapshot of the View Results Tree

Performance testing needs a more advanced kind of logging with graphs and statistical data. The component Aggregate Graph is very useful. During the performance run you can monitor all kinds of statistical data like the # samples, average, median, 90% line and min / max. All this per request and total average. The data can be written to a file (plain text, csv or xml) and after formatting can be used to create a performance report.

You also have the possibility to view a graph of the statistical data. Select you column, enter the width, height and titles and click the Display Graph button. Of course, you have again the possibility to save your graph or table data to a file.


Aggregate Graph

Name: Aggregate Graph

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Median	90% Line	M		Error %	Throughput
201_Startpagina	6	286	191	778	[illegible]	[illegible]	0,00%	2,1/min
202_Zoeken	6	977	623	2704	524	04	0,00%	2,2/min
203_Zoek_Pagina	6	516	540	686	353	686	0,00%	2,2/min
204_Detail_Scherm	6	297	206	774	193	774	0,00%	2,2/min
TOTAL	235	379	269	663	137	2704	0,00%	1,1/sec

Statistical Graphs

Title for Graph

Max length of x-axis label

Width 700

Height 400

Column Median

☐ Save Table Header

Figure 4 - Snapshot of the component Aggregate Graph

Conclusion

Need to test your web service, database, FTP- or web server? Both performance and functional testing? Try JMeter. It is free, very intuitive and has all the possibilities you need to automate your work. Being open source is another big advantage of JMeter. You can download the source and make modifications to it if you like. It is also very handy to have a direct contact with the developers through a mailing list.

Tip: Combine JMeter with Badboy (http://www.badboy.com.au/) to make it even more powerful! JMeter doesn't have record & playback functionality. Badboy is the solution. Record the flow in your website, export the recording to a JMeter file, modify it to your needs and use JMeter to test the performance of your site.

3 FREE Test Automation Guides to help transform your testing
Implementing Test Automation Tools - Describes a proven implementation framework for test automation.
The Test Automation Business Case -A guide to help you write a winning business proposal for test automation.
The Test Automation Check List - 59 essential questions to ask before you take another step with test automation.
Request these 3 FREE essential guides on software test automation TODAY! (registration required)

Requirements-Based Testing - Encourage Collaboration Through Traceability The root cause of 56 percent of all errors identified in projects are introduced in the requirements phase. Reduce your risk by implementing formal requirements management practices and requirements-based testing (RBT) with a goal to increase efficiency, reduce the risk of failure, and improve overall software quality. Learn how application lifecycle management can help your QA team implement RBT, providing insight into what to test, ensuring comprehensive test coverage, establishing traceability between requirements, test cases and code, managing the resolution of defects found during testing, and measuring project readiness through reports and dashboards. Get your copy of this MKS white paper now! (registration required) http://www.mks.com/mtmr_rbt

Advertising for a new software 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