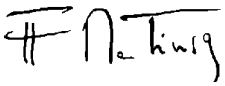
METHODS & TOOLS

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

Winter 2010 (Volume 18 - number 4) www.methodsandtools.com

Using Experts to Stop Serial Developers

In the television world, experts track criminals using DNA found on the crime scene. Luckily we don't have serial killers in the software development world, but we have what I will call "serial developers". If I had to "profile" them, I would say that they are often young, having recently completed their software development education, or worse with no formal developer training. They love programming, which means that they like writing code. Testing is not their favorite activity, but this is not a problem because they are confident about their code. Anyway you don't really have time to test in "real" projects that are behind schedule. You prefer to add more features to the product. Those features that ungrateful users will name bugs. Why are they stepping out of the "best case" scenario that I use to check my code? Don't we agree that users are the most important nuisance for software developers? If this description could seem rude to you, I must confess that I had part of these characteristics in my early years as a software developer. Some people evolve in their vision of what professional software development is, as I hope I did, but some people keep the same habits. They jump quickly from new projects to new projects, often leaving to other people the duty of polishing, maintaining and evolving their wonderful code. They will never know about the mess they can leave behind them. Maybe we are just too dumb to understand their code? To rephrase a famous Churchill quote about democracy: individual responsibility is the worst way to produce quality software, except all the others that have been tried. Unfortunately, in a lot of software development shops, performance is measured in lines of code and does not include any quality criteria. Personal control activities like peer reviews or pair programming are not well adopted and accepted. Developers have to be educated to the long-term consequences of their coding activity. Experience sharing and transmission could take different forms as the testing dojos article in this issue suggests. Education is certainly the best tool you can use to improve the quality of code. As in the TV world, you can also use tools to control software delinquency. Tools like Checkstyle controls coding standards. Potential problems, duplicate content or code complexity can be detected by static analysis tools like PMD. All the results of such tools can be aggregated in global tools like Sonar that will produce a dashboard showing the quality status of your projects. All the tools mentioned here are open source, so their adoption is not a cost issue. Using a tool to produce a static analysis report of some code is an excellent starting point to discuss how you can improve coding practices. Thanks for reading Methods & Tools in 2010 and I wish you all the best 2011.



Inside

Non-Functional Requirements: Do User Stories Really Help?

Rachel Davies, Agile Experience Ltd, UK
http://www.agilexp.com

Agile software development relies on bringing business people and developers together to deliver better software. We understand requirements by building working software and seeking user feedback on whether the latest solution meets their needs. This enables us to deliver business value early and to improve software in subsequent development cycles.

Agile teams focus on identifying user facing features that can form the basis of incremental deliveries. Often these are expressed as User Stories, slices of functionality that enable a user to achieve a specific goal. Developers work closely with stakeholders to understand what user stories must be satisfied by the product that they are developing. A flaw in this approach can be that users don't mention non-functional requirements and developers don't push to understand what quality attributes the software should satisfy.

So how does a team make sure they don't lose sight of "non-functional requirements"? Are user stories of any use in making these special kind of requirements visible to the team? This article explores how teams applying agile techniques go about resolving these concerns.

Types of Non-functional Requirements

Let's start with what people mean when they talk about non-functional requirements. Basically, non-functional requirements relate to qualities of the system that cut across user facing features, such as security, reliability, and performance. Non-functional is probably a bad name because it sounds like these requirements are intangible, simply properties of a static system. However, these requirements do affect the function of the system and it is possible to design tests that these qualities are present.

The difference from functional requirements is that these qualities must be present throughout the system rather than delivered in one-shot like a user facing feature. Alternative terms for non-functional requirements are "constraints", "quality attributes", "quality goals" and "quality of service requirements" but let's stick to calling them "non-functional requirement" (NFR) for now. So they have to be considered in every development cycle and factored into our test strategy.

NFR can be grouped into:

1. Execution qualities, such as security and usability, which are observable at run time.
2. Evolution qualities, such as testability, maintainability, extensibility and scalability, which are embodied in the static structure of the software system.

Agile teams need to consider both categories of NFR so how do we ensure that these don't get missed?

Taking Responsibility

When asked for user stories, stakeholders often forget to mention the NFR assuming that these will simply be there. This may be a fair assumption, if the organisation employs skilled software developers then surely they should always build software that runs well and is built to last.

Unfortunately, the team cannot telepathically know what the business needs and when pressure seems to be about delivering features consideration of NFR gets neglected.

I notice that developers who have been working on traditional waterfall projects are used to requirements being handed over fully specified and consider that their job is simply to build the requirements presented. On waterfall projects, NFR will have been considered upstream by architects and designers and already been factored into requirements presented to the development team. On an agile project, we can't make these lazy assumptions. As Dave Nicolette says "Many development teams don't grasp the fact that accountability for getting the requirements (all the requirements) right falls to them." [1]

An agile team has to take a more proactive role regarding requirements. User stories are not simply mini-requirements specifications that are handed over to development fully formed. Writing stories on cards is a very interactive process that involves the technical team asking questions about acceptance criteria that includes NFR. We must take responsibility for flushing out NFR so my first tip for capturing NFR is to actively start looking for them!

Consider Non-functional Requirements from the Start

Agile teams usually take an evolutionary approach to architecture and design. Agile approaches do not specify what roles need to be on the project because this depends on what kind of system you are making. Make sure that you engage with technical stakeholders in your organisation, such as architects, user experience designers, and operations teams. These people can help an agile team spot NFR that are not captured in your user stories.

The development phases is a short time in the lifetime of the product, most software systems are in use for many more times longer than they take to build. Make time to understand the user context better by learning about the different types of users and the situation in which they will use the software. Start conversations about the operating environment and user expectations early. Talk about the capabilities of the system, like quality of service and longevity, at the start of the project. You can do this by organising a vision workshop and create a project charter that includes system non-functional goals as core success criteria.

You can also hold a session with the team to brainstorm risks to consider what are the things that can go wrong once the system is live? Many potential risks can be mitigated by identifying NFR to prevent failures. Another benefit of linking NFR to risks is that you can identify costs of missing NFR and this information can be used to prioritise work on NFR against user facing features. For example, how much will it cost the business if the site goes down? How many users will we lose if the system is too slow? Estimating such costs helps to show that work on the system NFR is often just as valuable to the business as the more obvious value generating features.

Using Stories for Non-Functional Requirements

How does an Agile team factor work on NFR into their plans for incremental releases? Is it helpful to try and describe these as user stories or not? I've talked to many experienced agile teams and found different approaches advocated. Let's take a look at some of the pros and cons.

Lots of teams use a user story template like this:

As a <user role> I want <goal> so that <business value>

Some teams like to use this template for everything that they work on including NFR because that's a consistent format.

The beauty of this approach is that all requirements look the same, can be understood by everyone including non-technical folk, and therefore have the same visibility as other stories. What's surprising about this approach is that normally elaborating requirements as user stories help a development team understand user needs better. However, user stories expressing NFR don't necessarily help the development team understand what needs to be build better. Instead, shaping NFR into user stories helps people outside the team understand the benefits better.

The advantage of managing NFR this way is that they are visible to stakeholders. Unfortunately, this can also become a disadvantage, if these requirements may be seen by business stakeholders as optional or "nice to have". It's important to include the "so that" part to make it easy to see why such NFR stories are important from a business perspective to help ensure that they don't get de-prioritized simply because they are not understood.

Other teams keep it simple and simply factor NFR into acceptance criteria for affected user stories. This effectively makes work on NFR invisible. The odd thing about doing this is that any stories which imply significant architecture work have higher estimates than others but because the architectural piece is below the surface it the estimate belongs to whichever story that gets worked on first. This can be difficult to explain to stakeholders.

Another approach is to introduce "technical stories" to cover anything that needs to be built or restructured but is too difficult to explain to business people. These story cards are typically free format and expressed in language that only the development team understands. The developers are trusted to include only those technical stories that make sense often working within an allowance of such stories per iteration. The benefit is that these don't get de-prioritized by business people and yet these stories explicitly appear on the team board and progress on them gets discussed in daily stand-up meetings. This approach is often popular for teams working on legacy systems with many complexities that need to be cleaned up and made testable. Teams at BSkyB have even called this allowance for technical stories a "Quality tax."

One worry with this approach is that this excludes the business from deciding what the project budget is spent on. Henrik Kniberg's advice is "Try to avoid tech stories. Look hard for a way to transform a tech story into a normal story with measurable business value. That way the product owner has a better chance to make correct tradeoffs." [2]

Testing Matters

Sometimes checking for NFR requires specialised “expensive” testing so you may need to plan in additional testing periods. Watch out that these test periods don’t become an excuse to defer work on NFR as this may put release dates in jeopardy.

Agile teams cannot rely on manually checking that user stories meet agreed acceptance criteria. Look for ways to automate system tests so that these can be run daily (or more frequently). Build checking that NFR have been addressed into the team definition of Done so that these remain visible for the whole team whatever feature they’re working on.

Summary

Waterfall has got many developers into bad habits. When requirements come from someone else and someone else tests my work, I don't need to know the environment that my software will be running in. Agile developers need to challenge this attitude and work with a variety of stakeholders to understand the world that their software will exist in and bear that in mind for each story that is deployed.

There is no magical agile practice that helps you uncover NFR. The first step is to take responsibility. NFR can be represented as User Stories if the team finds a that this helps to keep these visible. However, be aware that surfacing such stories may create issues around the priority of work done on them against more obvious features.

References

1. “Managing non-functional requirements and enterprise standards” blog by Dave Nicolette, http://www.davenicolette.net/

2. “XP and Scrum from the Trenches” by Henrik Kniberg, http://www.crisp.se/ScrumAndXpFromTheTrenches.html

Further Reading and Watching

"Non-Functional Requirements: Do User Stories Help?" Video and PDF, Rachel Davies presentation at DevOpsDays 2009, http://www.devopsdays.org/ghent09/programme/

Testing Dojos

Markus Gaertner, markus.gaertner @ it-agile.de
it-agile GmbH, http://www.it-agile.de

Programmers use Coding Dojos[1] to exercise their programming skills in a group setup. Deliberately they get together to pair program and get feedback from peers. But how do testers train their skills? Testing Dojos provide a similar setup as Coding Dojos, but help testers to practice their individual skills in a group.

What is a Testing Dojo?

A Testing Dojo is a meeting where testers come together to work on a testing challenge. The challenge can consist of testing a product, generating test ideas for a particular software or even exercising bug reporting. The challenges will use mainly Exploratory Testing, but learning a new test automation tool is also a very nice mission.

A group of testers may decide to get together on a Lunch & Learn meeting each Friday and test an application from the Internet. The team members take turns facilitating the session and coming up with a mission and a product. The facilitator is responsible for the session. She can pick an application from the company or one that she uses regularly during her free time. It could even be one that she came across on the Internet. A mission provides the goal for the testing activities. Setting up a mission for a particular application might be a bit more difficult. That's why we will examine testing missions separately later.

Testing Dojos take testing to a safe environment without schedule pressures and deadlines. Testing Dojos are a way to train testers new to the profession or the company in a collaborative manner.

Equipment

For a Testing Dojo you should have some things in place:

- a meeting room large enough for the group
- access to a computer
- a video projector so everyone can see what's happening
- pen and paper, a flipchart or a whiteboard to take notes

A meeting room with a computer and a projector are available in most companies. If the meeting room lacks a flipchart or a whiteboard, it is easy to get pen and paper for note taking. You will get more benefits if the notes from the session are clearly visible to everyone in the room, especially for a mission to learn about note taking. You can look at tools like Session Tester[2] or Rapid Reporter[3] to take session notes directly on screen.

Roles

During a Testing Dojo, each person in the room always fills one role. There is of course the role of the tester who has the power over the keyboard and who interacts with the software. Then there is the recorder's role, which takes session notes and makes sure to get reproducible steps noted down. The observer watches the performance. He thinks about suggestions for improvements and gives observations about communication in a paired setup. Finally, the

facilitator makes sure that the rules of the dojo are followed.

Tester

As the tester has the control over the keyboard, he interacts with the program. It's crucial for the tester to expose his testing ideas about what to try or not to try and describe his mental model to the audience.

Recorder

The recorder takes session notes about the activities. In a single tester setup the tester is simultaneously the recorder and he has to take care about his own session notes. In a paired setup there is a dedicated person for this.

Observers

In a Testing Dojo there are one or more observers, depending on the size of the group. The observers take notes about the process. These notes include the interactions between tester and recorder in a paired setup or spoken thoughts and test activities in a single setup. Observers should monitor the testing activities and the interactions between the pair, rather than watch the screen. Additionally, observers should try to keep track of one or two points, rather than trying to observe everything. If you are neither testing nor recording, you are not taking a break. As an observer you must keep engaged in order to follow what happens.

Specific things to observe might be test heuristics[4], test framing[5], oracles, coverage ideas, management of focus, conversation, monologues, pauses in talk, pauses in testing, eye contact. The facilitator might want to hand out some common oracles [6] or mnemonics[7,8] for the participants.

Facilitator

The facilitator picks the mission for the dojo. She enforces the rules like switching timeframes. At the end she moderates the feedback activity. The facilitator could also take on the role of the tester, recorder or observer, but her main responsibility is to facilitate the dojo. A junior tester can take on the role of the facilitator in order to learn the skills needed to lead a group or meeting.

Mechanics

A Testing Dojo starts with the facilitator introducing the rules. The facilitator provides a mission and clarifies the structure of the dojo. If you run a Testing Dojo for the first time, you could consider providing a first focused test that gets the group going, rather than leaving it too broad.

The session facilitator should change so that everyone gets the opportunity to lead a small group of people.

Any testing can be done by a single tester in front of the computer or in a paired setup.

The missions vary between testing a product, evaluating the usage of the following tools or using a new approach to check if we could incorporate it into our testing process.

When the team has run many Testing Dojos, the introduction tends to get shorter since the facilitator has to explain less of the standard dynamics.

Single tester

In a single tester session, the person with access to the keyboard takes on the role of the tester and the others fulfill the role of observers. On a previously agreed upon time the tester is replaced by another participant from the audience. Five minutes seem to be just enough for this, but you can try different timings based on the size and skills of your group. The new tester continues to follow the mission tackling the product under test. When the individual tester gets stuck, he can ask for support from the observers. Otherwise observers have to remain silent while watching the performance.

For instance the facilitator has picked the latest application from the company and decided to explore it. A tester unfamiliar with the software is asked to begin for five minutes. Right after starting the application, she faces a problem because she does not know what it does. After a few hints from the observers about the documentation, she continues to explore the software, but does not quite understand it. After the five minutes have expired, a tester more familiar with the application starts, coming from the group of observers. He shows some of the features and explores areas of the product. He provides the first tester with some new insights about the application, as well as the approaches used to test it. A third tester then takes charge and explores the product from a completely new perspective.

The tester must explain every step of her thoughts for the observers to follow the individual actions. As a rule of thumb, the more the tester speaks, the better it is for novice testers in the audience. More experienced testers need less explanations of a particular motivation or applied heuristic.

Paired session

In a paired session, two participants sit in front of the computer. The tester is working on the keyboard, while the recorder writes down the test ideas and discovered bugs. After a previously agreed timeframe, between five and ten minutes, the tester goes back to the group of observers. The recorder takes over the role of the tester and one of the observers becomes the new recorder.

During a paired session the tester and the recorder in front of the computer need to clarify their steps so that everyone from the observers understands what they're doing, and more importantly why they are doing it. They should at least talk as much as they test and probably talk more than test. The pair is allowed to involve other participants only when it gets stuck just as in the single tester setting.

The paired setup is similar to a Randori Kata in a Coding Dojo. The people in front of the keyboard need to make their steps clear to the audience. Over the course of the time-box, they explain the model they got from testing the application. This knowledge is then transferred when the pairing partners switch. Since all observers watched and listened to the particular steps performed, the next partners will have similar knowledge about the product. Over the course of the session, each participant takes notes and tests.

Pairs switching can be decided using a round robin style or by picking the next volunteer for a larger group. In round robin switching, each participant should get into the role of the tester and the recorder at least once. The size of the group will influence how much time you schedule for

the session. If you have one or two testers who feel uncomfortable to expose their testing steps to others, you can make testing and recording optional.

To make the activity more fun, you can include a ceremony activity for pair switching like a handshake when the tester leaves or a Karate-style bow when a new recorder joins. You can also bring in special things if the team found a bug, like a high five.

Suppose that John and Susi are the first pair. John decides to pick up the keyboard and Susi begins with recording notes. They take the application and discuss which areas to focus on. The session facilitator starts a timer for five minutes, informing the pair when they have two and one minutes left. After getting a brief charter up, John and Susi start to test the application with the first item on their list. They make pretty good progress. The time also passes very quickly. John goes back to the observers after the facilitator announced that the time is up. Susi now takes the keys and Paul joins her. Paul raises a question about one area that he just saw while John and Susi were performing. Susi gets back to this area and Paul asks some questions that eventually result in Susi doing some follow-up testing.

Missions

This section takes a brief look into possible missions for a Testing Dojo. Testing missions provide the goal for any testing activity. Testing missions can vary between “Test this”, the evaluation of tools or learning new approaches.

Test this

The classic mission involves testing an application. The variety of applications includes open source programs, commercial software available in your organization or even your company's latest product. Such a session can end up as a bugfest. The facilitator needs to check if the mission is doable before the session starts. The company's firewall may block some content on a web page or it might be cumbersome to test a web page with unstable network connections.

This type of missions has many different forms. The facilitator can focus the session on a particular aspect of the application like usability problems. The audience could get to know how to learn and model a new product without thorough documentation.

You could also pick test automation, though this needs more planning and preparation from the facilitator's point of view. The facilitator should provide some initial examples of automated tests and then ask the participants to extend them. Usually during a short period of a single hour or two, automating test cases for a new application seems to be not as productive as an exploration session conducted with mostly manual tests.

Evaluate tools

A mission to evaluate a tool could use mindmaps for test ideas or try out a particular test tool for the whole session. In such a mission, the product under test is usually the tool itself, but you can run it also for a common program that you test at work and compare the results directly with your daily work. Evaluating tools can help to decide if you want to learn more about them.

You could evaluate an open source application for writing test documentation similar to the company standards or assess several free online services such as URL shorteners to be incorporated into the next product. Yet another mission could be to evaluate a new bug tracking system and provide management with information regarding the capabilities of the tool given the company's context and culture.

Learn new approaches

There are many testing approaches to try out. You could focus the mission on some particular mnemonic like FCC CUTS VIDS[9] or use soap operas[10] to generate test ideas. Like tools evaluations, this type of mission aims to try out and learn about new approaches. After these sessions, the whole team will have made some experience and can make a more informed decision about the usefulness of the approach.

Reflection

Take some time after each session to think about it and share observations. To keep the discussion focused on suggestions, you can take turns with stating positive things from what you saw. After that, each participant can provide specific suggestions of what can be improved. Try to fill in the template "I suggest to…", and remind participants to stay positively focused at this. It's easier to accept feedback in this manner.

On a meta-level you should also think about things to change after each dojo. For example you can modify the switching time from five to six minutes, drop a rule or try something new in the next session. The group should make the decisions.

After the reflection, the team votes for the next facilitator, the time and place for the next dojo. The new facilitator becomes responsible to schedule and organize the next session.

Putting it all together

Testing Dojos help your team gain a shared understanding of their approaches to testing. The collaborative setup allows sharing individual approaches quickly. New testers can directly see how a more senior tester would tackle the program, while a more senior tester can get new insights from the fresh perspective of the rookies. A Testing Dojo conducted with project managers and programmers can bring transparency to the magic thing that's called testing. This surely aids in the team building process, since the whole team gets to know how a skilled tester works.
Based on your team and company culture, you can play around with conducting a Testing Dojo each other week, running a Coding Dojo between them. With this alternation you make sure, that you get a balanced approach to your deliberate practice. In the end, programmers attending a Testing Dojo might even find out the value of testing.

You can use Testing Dojos in your local testing user group. Check the Internet if there is already a public Testing Dojo in your neighborhood.

A rich variety of Katas exists for Coding Dojos. There is currently not an equivalent resource to pick testing missions from. One source of inspiration is the catalogue of testing missions from Weekend Testing sessions[11]. Blogs about testing are another source of information. You can also simply ask your favorite search engine on the Internet about a testing challenge. As more teams use Testing Dojos, the collections of possible missions might start to grow.

Happy testing.

Special thanks to Tiina Kiuru and Michael Bolton and all the attendees of the Agile Testing Days 2010 Testing Dojos on their early feedback.

Bibliography

1. Coding Dojos, http://codingdojo.org
2. Session Tester, http://sessiontester.openqa.org/
3. Rapid Reporter, http://testing.gershon.info/reporter/
4. Test Heuristics Cheat Sheet, http://testobsessed.com/2007/02/19/test-heuristics-cheat-sheet/
5. Test Framing, http://www.developsense.com/blog/category/test-framing/
6. Testing Oracles, http://en.wikipedia.org/wiki/Oracle_%28software_testing%29
7. Testing Mnemonics, http://www.qualityperspectives.ca/mnemonics.html
8. The Power of Mnemonics, http://curioustester.blogspot.com/2009/10/power-of-heuristics-and-mneumonics.html
9. FCC CUTS VIDS test heuristics, http://www.michaeldkelly.com/archives/50
10. Soap Opera Testing, http://www.logigear.com/logi_media_dir/Documents/whitepapers/soap_opera_testing.pdf
11. Weekend Testing, http://www.weekendtesting.com

Process improvement, The Agile Way!

Ben Linders, Senior Consultant, info @ benlinders.com
www.benlinders.com

Business needs for process improvement projects are changing. Organisations expect faster results from their investments; they want their improvement projects to adapt to and follow changing business needs and be more engrained with the organizational way of working. The agile way of working, used more and more in software development, contains several mechanism that support these business needs. So the question is: Could a process improvement project be performed in an agile way and what would be the benefits?

In this paper I start by looking back to my first software development project. I managed that project in a way that would now be called agile, to be able to meet the needs of my customer and of the organization. Next I'll give a brief description of process improvement, and of agile; just the basics needed to understand how they can be melted into an agile process improvement approach. Then I'll go into the reasons to do it process improvement in an agile way, and the benefits that can be expected from it. I will discuss a distributed process improvement project that has been managed in an agile way, to share the learnings and benefits. Finally I'll describe some "golden rules" that help to improve agile working along the way, and to become even more effective in it.

My first project: Working Iteratively

The first project that was assigned to me was managed iteratively by me. This was not demanded by the line manager he just wanted me to finish the project as soon as possible, and deliver a product; it was up to me how to do it, and to come up with a plan. The product to be developed was software for a CNC milling machine, for milling of a so called pocket. A pocket is defined by a mathematical formula that specifies the outer boundaries (often called contour) of the area that is to be cleared with the CNC milling machine. The product to be developed is a complex mathematical embedded software solution that calculates tool paths. It has to be integrated in the complete system (hardware and software) that is used to program and control a CNC milling machine. The product was developed and maintained by an organization of +/- 60 professionals, organized in product teams that each were responsible for a part of the product. The organization combined development and maintenance work, with projects for developing new functionality, and problem reports for handling maintenance and smaller product updates. Previous projects that had tried to develop the new software for pocket milling failed to deliver a stable working product, or were stopped since they weren't able to deliver. So the customer had lost its trust in the product, and we were aiming to regain it by developing and delivering in iterations. Working iteratively would also help us break down the complex technical problem, and to deal with the technical risks early in the project.

We learned from the previous projects that the biggest difficulty was the huge set of different types of pockets that the software should be able to handle. Different algorithms had been implemented, but none of them was able to solve all the possible pockets. We decided that the purpose of the first iteration was to implement a basic version of the algorithm for calculating tool paths. Purpose of this delivery was twofold: First to see if we could at least provide a solution for the "easy" pockets, and second to regain trust of the customer and get feedback from him. The software delivered was not running on the target (embedded) platform but on a PC, and it only had a very basic user interface. Also it could only handle a very limited set of possible pockets that could be milled. We demonstrated the software to our customer when he visited the development site, and he took the software back home and used it on his PC to test

several pockets. His response was that the software actually worked with most of the basic pockets, but he found some situations where it didn't function correctly. Since he could clearly describe when it failed and how, we were able to reproduces the problems, and solve them within a couple of days.

We had a next iteration where we extended the algorithm, and delivered a second version of the product which supported several new pockets. The delivery also included solutions to the problems that were found on the first version. Again we gave a demo to our customer, where he actually tried some of the pockets that failed in the first version, and saw that they were handled correctly now (talking about an interactive demo, it was!). Again he tried the software on his own PC, and confirmed that it already provided a solution for a lot of the pockets that it was required to solve. In a discussion with the customer we were able to define which pockets should be implemented next, and what would be the priority to deliver solutions for those pockets. We also identified several pockets that were possible in theory, but that would never be used by the end customers of our application, so we did not need to provide solutions for them. Some of those pockets would have been very difficult to solve (and would have cost a lot of money to develop and test), so by clarifying our requirements with our customer we saved a lot of time! We continued to deliver iterations, until the customer decided that the available functionality was sufficient to include in the next release of our product. The project was evaluated and finished, and our customer was pleased to see that the functionality that he had been waiting for years was there. Our project took about a year, less then any of the earlier projects, and it actually delivered value to our customer.

So what where the advantages of working iteratively? First, we regained our customers trust, simply because we managed to deliver. We got lots of useful feedback, from our customer and from fellow development groups. The feedback mainly helped us to clarify our requirements and get a better understanding of what our customer needed, and what he needed first, and what he didn't need at all. It also helped us to improve the quality of the software.

One small thing that I didn't mention yet, the project with iterative development that I am talking about was done in 1991. There weren't that many books at that time on IT Project Management, let alone on iterative development, demonstrations and frequent deliveries to customers (which are now practices of agile methods). There was hardly anything published on starting with a first set of requirements, and clarifying and adapting your requirements during the project, preventing to spend time on requirements that turn out to be wrong or unneeded (which Lean calls waste nowadays).

Iterative methods like RUP and Scrum weren't invented yet; the only iterative method that I was aware of at that time was Evolutionary Delivery [Gilb 1988]. I considered working iteratively as the logical thing to do. Though working iteratively was new to the organization, it made sense to our customer, the management and the team that I worked with, so they allowed me to try it out.

Distributed teams

After having delivered some first iterations of the product that focused on the mathematical algorithm, the need came to develop the user interface for the software. There were some experienced software developers who could assist the project; however they were living at the other side of the Netherlands. We defined a way to collaborative develop the software, while working on 2 locations. First we agreed upon the architecture, and the main interfaces between the algorithm software and the user interface. We also agreed upon some technical things, like the operating system and programming language to be used, the tools used to compile and build the software, and some basic configuration management aspects like file names, version numbers and how the files at the various development sites would be synchronized. Finally we drew up a short list of design and programming rules, mainly because we wanted to be able to read and understand each other's code.

The teams had one fixed day every week when they worked at the same location. At that day, code was integrated, rebuild, and jointly tested. Also there were technical discussions, and members of the team peer reviewed each other's code. The other days the team members worked on two sites, where they occasionally contacted each other when any issues or questions came up. Since they knew each other quite good, such contacts were brief, to the point and very effective.

What was the advantage of working with distributed teams? The biggest benefit was that we were able to have professionals on our team with the right expertise (though working remote most of their time), in stead of having professionals that were working locally but didn't have the right knowledge and skills. This saved us a lot of time and money, and increased the quality of the product that we delivered. Having a fixed day where the whole team worked on one location helped us to communicate and synchronize and assure that we had working software every week. Remember this is 1991, practices like continuous integration did not exist, in fact the only connection that we had between the sites was a dial up modem (there was no publicly available internet) which was much to slow to synchronize all source code, and there were no tools to support synchronization. Also our programming rules turned out to be helpful when we peer reviewed each others code, the discussions where mostly about how the problem was solved in software, and less about layout and coding styles, which made the review very effective.

What did I learn from my first project?

In my first project I clearly saw the benefits of working iteratively. I learned that we were able to deliver quicker to our customer, and deliver only what was needed; not wasting time on things that were not required. Also the early customer feedback helped to deliver the right quality. My team members liked to work this way, and (also important since this was my first project) my manager was happy. This all was confirmation for me that this was an effective and efficient way to develop software. My main learning of working with distributed teams was that is was actually possible to do it. You need to arrange things to be able to work smoothly on multiple sites, but then you can work together. It is important though that the team members know each other, so initially people have to work on one site, or they need to travel between the two sites to work together and build up a relationship.

As far as I can recall, all of my software development projects have been iterative, and lots of them had distributed or dispersed teams in some way. Even when there was no possibility to do early deliveries to customers, we develop and tested the software in iterations to have early internal feedback, and to have working software early and often in the project. This way of working has a lot of similarity with what is nowadays called agile.

So when I started to work as a quality manager, and later as process improvement manager, working iteratively was a logical choice for me. And I was confident to work with distributed or dispersed teams, knowing the benefits it could bring and being aware of the preconditions that make it possible. But before I describe how we applied this iterative and agile approach to process improvement, let me first describe what process improvement is, and give you some insight on the agile principles that were used in our approach to process improvement.

Process Improvement

Given that software development is still a young and immature profession, there is a need to continuously improve the development and management of software. Software Process Improvement (SPI) is one way to arrange this. SPI is based on the assumption that there is a process that describes how the software is developed, and how this development is managed. By improving the underlying process, the quality of the software that is the result of it will improve. It also assumes that the process, which governs how the software development is managed, can be improved to deliver software on time and within budget. Since the principles describes in SPI can and have also been used in projects that deliver hardware, or presentation and documents that can be used to improve an organization, I use the more general term process improvement in this paper.

Many organisations use the Capability Maturity Model – Integration ® (CMMI) [CMMI 2006], which a collection of best practices. The practices are arranged into process areas. Each process area has defined goals that need to be satisfied, and a set of practices that can be implemented to reach the goals. There are two representations of the CMMI model: Staged and Continuous. The staged representation assumes that process improvement is implemented using 5 maturity levels, where each level (except level 1) has a set of process areas. Process areas of higher level assume that the process areas of a lower level are in place; staged implies that the organisation is improved in steps from maturity level 1 to 5.

The continuous representation provides an organization to pick those process areas that would bring the biggest benefits, and implement the practices from those process areas to improve capabilities of the organization [Linders 2001]. This implies however that an organization will have to have some insight in the process areas and the potential benefits, to be able to make a good choice of process areas to implement. The CMMI suggests doing a zero-assessment to determine the current capability level of a process area, and then define the wanted capability level of each process area and implement the changes to bridge the capability gap. Many organizations starting with process improvement do not have sufficient insight and skills to take this approach. To help them to get started with the continuous representation, an additional approach was developed, called CMMI Roadmaps.

CMMI Roadmaps [Cannegieter 2008] are a goal driven approach to select process areas from the CMMI model. There are 5 pre-defined roadmaps of CMMI process areas. A roadmap consists of 4 to 6 related process areas. CMMI Roadmaps help organizations to focus upon a specific solution area. The roadmap approach combines the strength of the staged and continuous approach, by helping an organisation to get started with CMMI and reap benefits, while selecting those process areas that will most probably give the biggest benefits.

Visibility is crucial for reaching results with process improvement [Linders 2006]. Process improvement is a learning process, where professionals change their way of working and reflect to see if this leads to improved performance. By making the aims and the results of process improvement visible, professionals can exchange experiences and learn from each other. Visibility enables alignment between process improvement and the business goals, which is vital to gain and keep commitment in the organization. Visibility also shows success, and since success breeds success this helps to increase adoption of improvements. Later in this paper I will go into how the agile approach to process improvement supports visibility, thereby increasing the performance improvement of the organization.

Agile

The term agile was coined by group of people experienced in software development, when they gathered in 2001 to define the manifesto for agile software development [AgileManifesto]. As stated in the manifesto, they are “uncovering better ways of developing software by doing it and helping others do it”.

Agile is not a specific method, but includes a broad set of methods and techniques that subscribe to the values and principles of agile. Well-known agile methods are XP and Scrum. But also a method like Evolutionary Development or EVO [Gilb 1988] is agile, even though it was there long before the term agile was invented.

The agile manifesto contains values, one of them is “to value individuals and interactions over processes and tools”. Process improvement as we saw earlier puts much emphasis on the process as the base for developing software. At first sight, agile and process improve might look incompatible, but that is certainly not the case. As the agile manifesto states, it is not that that they do not value the right statement, but they value the left more. So as long as processes help professionals to interact effectively and efficiently, they can certainly be agile.

Agile Process Improvement

Let’s go back to the question if we can do process improvement in an agile way? First I’ll describe the reasons why you want to become more agile, and the benefits that it can deliver. Next I’ll go into detail how agile process improvement was actually done. I’ll finish by looking

back if the agile approach for process improvement has paid off, and will show of "golden rules" that were used to deploy the agile approach and to further improve along the way.

Why Agile Process Improvement?

There are a couple of reasons to become more agile when doing process improvement. From a business point of view it is more and more important to be able to adopt process improvement programs to the changing business needs, and to be able to deliver more value to the business. This included making the value visible, which supports discussions on what value has been brought already, and what value can be expected from process improvement. Another reason to become more agile was to engrain process improvement in the way of working within the organization. Many organizations have or are adopting agile development methods, and use techniques like retrospectives to continuously improve on a team level; it feels natural to adopt similar mechanisms for process improvement programs. Finally, since there are almost always limitations in lead-time and money (available hours of the team and of the organization that need to adopt the changes), you have to work as efficiently as possible. Agile helps the improvement team to deliver quickly, and deliver maximum value with a limited budget. So altogether there are sufficient reasons, from a business point of view to adopt an agile way of working for process improvement.

Additional there are several benefits that an agile approach to process improvement can deliver. Firstly, agile can improve the collaboration between the process improvement team and the stakeholders. Agile iterations enable frequent interactions between those involved in process improvement, involving the customers of process improvement much more into the change process as the "traditional" process improvement approach. The agile approach also helps to focus on deployment of the processes in stead of making large process documentation, based upon the second value of the agile manifesto: "working software over comprehensive documentation". Finally by working agile themselves, the improvement team gets a good understanding of agile which can be used to better support to teams in the organization that are also working agile, or making the transition to agile. Basically it's eating your own dog food, which helps you to get insight into the consequences and the obstacles of changing towards agile, while at the same time getting the benefits of it. The improved collaboration, focus on deployment, and personal experience with agile of the process improvement team leads to a bigger impact in the organization, and therefore to increased organizational performance and business results.

Project: Improving Process Management

The agile approach has been used in two process improvement projects that I presented at the SPIder conference in 2009 [Linders 2009]. One project concerned the improvement of process management activities within a business unit of a multinational company. This project was done with a dispersed team, consisting of operational development professionals from R&D centres around the world. The other project was executed in one of the R&D centres. Purpose of the second project was to do selective and quick process improvements, to continuously improve the performance and results. This paper goes into the first mentioned project, and describes how we have used the agile approach, and what the benefits have been for the project, team members and the organization.

The purpose of the process management project was to align the various development and delivery processes that were used by different R&D centres within a business unit. Expected benefits were that by using the aligned processes, the business unit would be able to develop and release products quicker, thereby being faster to the market with new products with high quality.

Cost was also considered important, but by making the flow leaner and remove unnecessary activities, the cost would automatically go down. The product was ordered by the global business unit, and involved several R&D centres spread around the world.

The project used Scrum as method to coordinate and collaborate [ScrumAlliance]. Though Scrum is mostly used by agile software development teams, it looked suitable for our process improvement project. Scrum recognizes 3 roles, the Product Owner, the Scrum Master, and the Team. The manager responsible for processes within the business unit acted as Product Owner, his prime responsibility was to define and prioritize the user stories. The user stories described the services and products that had to be delivered by the Team. Given that this was a different kind of project, the products produced were mostly presentations, slide sets, and documents containing policies and texts for the Intranet or newsletters. Next to products there were also tasks that were done, like assuring that issues were discussed in the appropriate groups and meetings and that decisions were taken and communicated, and activities like arranging meetings and workshops to discuss and review materials that the team had produced.

The project manager took the role of Scrum Master, and adopted the project plan to reflect the agile way of working. A major step was to publish the project information on the wiki in stead of a word document, thus facilitating the discussions of project scope and approach using this wiki. Actually our wiki, which started as a project space grew to being one of our main communication tools, planning board, delivery platform and even demo space; we continuously evaluated our use of the wiki and found new ways to use it as a team working space.

The team consisted of professionals involved in operational development at the R&D centres. They were very experienced in process management, but most of them had never worked in an agile team before. Based upon my experience with agile I got the role as agile coach, next to my role as team member in the agile team from an R&D centre. In that role I helped my fellow team members on how to fill in the different Scrum roles, by coaching them on activities like the planning game, stand up meetings, and how to use the wiki. I also organised the retrospectives at the end of every iteration, and supported the team members to implement the improvements that were suggested in the retrospective into the iterations. All team members worked part time on the project, where the weekly availability varied from just a couple of hours to some days per week. Occasionally team members where not available, due to local R&D centre activities or travel. Since the team was aware that they were depending on each other to deliver quickly, we kept each other up to date on the wiki regarding our availability.

Having all Scrum roles assigned we started to use the Scrum principles to organise our work. As mentioned earlier, a backlog was set up with user stories from our product owner. We started with a limited set of user stories, which were extended whenever new products or services that the team needed to deliver where identified. We had a first planning game at one of the R&D sites, where all team members gathered with the Scrum master and the product owner to discuss and define a first sprint. We agreed upon the user stories to be included, and how we would do our “stand up” meetings, use the wiki to collaborate, and do the demo for our customers and the retrospective to learn and improve as a team. We decided to work in sprints of 3 weeks. Each sprint started with a planning game, and ended with a demo and retrospective, in which all team members participated. The demo was given to the product owner, and we also invited relevant members for our organizations to attend and give us feedback.

The sprints were done while the team members were working at their own R&D centre, so we worked most of the time as a dispersed team. At several occasions, members of the team met and worked together at one of the R&D sites. For instance, when a team member was visiting another R&D, they arranged to meet and work jointly on tasks. But most of the collaboration was done using the wiki and telephone conferences, which turned out to be a very efficient and effective way of working together.

The wiki was our task board, and the communication tool both within the team, and from the team (including the product owner) towards the organization. The project plan was a wiki page, which included the goal of the project, the way of working (our process), and contact information of all team members and stakeholders. Also the backlog was on the wiki. This created visibility for the improvement project; both in the communication within the team and from the team to the stakeholders. In emails and in newsletter, references were made to relevant pages on the wiki, we saw that many people clicked on the link and read the information on the wiki. This is different from project plans in for instance Word, which many people do not read when sent as an attachment.

For every iteration, a separate wiki page was made. Initially this page contained the availability of the team members for this iteration. Planning games were done by telephone conference, where the ScrumMaster continuously updated the wiki during the meeting, based upon the task breakdown and estimates from the team members. The user stories and engineering tasks were added, prioritized, and checked by the team which at the end of planning committed themselves to deliver the required functionality. The wiki was also our planning board, so when a team member started to work on a task, he marked this task on the wiki and added his/her name to it. Two times a week there was a “stand-up meeting”, were all team members dialed in and again used the wiki to discuss what they have done, what they will do next, and if there are any obstacles. Intermediate versions of products were posted on the wiki, and were reviewed by other team members which put their comments on the wiki. The last days before the demo, all materials were gathered from the wiki into a demo presentation. The demo consisted of a phone conference with presentation. Team members and the stakeholders either joined in from their workstation, or (if multiple people were at 1 development site) arranged for a conference room with a beamer and hands-free telephone connection.

It took a couple of iterations to tune our “definition of done”. Having a presentation or document made was not enough, so we decided that a task could only be done if the products had been reviewed and commented upon by at least one other team member, and the product was updated based upon the review comments. Of course all documents and presentations has to use the official templates and comply with the corporate standards, but this was something all team members were already familiar with.

Using the wiki has really given the team a boost. It was not uncommon that a document was updated and posted several times a day, using the comments that team members shared on the wiki. If possible team members responded within hours when a document was offered for review, also documents and presentations were shared so where this was possible team members updated the documents directly. If a team member saw the need to discuss a comment either with 1 or multiple team members, a conference call was setup. Since we also mentioned shortly on the wiki how a comment has been handled, it was also possible for team members to catch up (remember, we all worked part-time on this project), or to trace back how we had come to certain decisions.

Finally since out wiki task board always reflected the latest status of our tasks, there was no need for burn down charts. Working in this very disciplined way felt natural for the team

members; it was their solution to interact and collaborate in this dispersed team. It was also a very efficient and effective way of working, enabling the team members to combine the projects tasks with the other activities that they had to do.

Benefits of agile process improvement

Looking back there were several benefits from the agile approach as we used it in our global process management project. The main benefits that we saw were:

- Being able to deliver the right product with high quality, using frequent feedback
- Understanding the strengths & weaknesses of our processes, and the business value
- Alignment and streamlining of processes between several R&D centres
- Efficient ways for professionals to work together in a dispersed team

We were able to produce many documents, newsletter, slides sets and supporting material that the local R&D centres could use to align and improve their development processes. Though we have no hard figures, our impression is that it took us less time then usual to make them. An even bigger benefit was that the quality of the products delivered by the project was very high, and there was a strong commitment for the changes proposed by the project, due to the frequent planning game and demo meeting where we aligned with our customer and the stakeholders. Several R&D centres started to use the material while it was still under development to do their local process improvements. This helped them to start quickly, and come to results faster, and it also helped our process improvement project since we got feedback from the local improvement initiatives and were able to update our products along the way. All together the commitment for the process improvement initiative was high, both at a business unit level and at the local R&D centres.

Working intensively together with the business managers (our product owner and the stakeholders) helped the team to get a better understanding of the strengths and weaknesses of our processes from a business point of view. Together we developed an overall development & delivery flow, which identified the main decision and synchronization points in product development. At a lower level, local processes were linked into this flow, enabling sites to continue to use and further improve their way of working. An important result was how we managed to combine both traditional and agile development methods in the overall flow. Several R&D centres had migrated to agile; some were considering it; for those development sites it was very important to see how they fitted into the overall flow. Vice versa it helped functions on a business unit level, like product portfolio management and product release to work with the different R&D centres in a similar way regardless of the processes used, thereby saving money and being able to deliver earlier to our customers.

The process discussions also helped the team and the stakeholders to get a better insight into the value of our processes, and how much processes were actually needed? In some cases, processes were phases out since they were no longer to be used in the organization. This could be because there were similar processes from other R&D centres, and we decided to combine the multiple processes into one description which could be used throughout the business unit. In some other cases we found processes that were so detailed that it was almost impossible for people to understand and use them. The proposed solution was to streamline process documentation into two levels. At the high level, process documentation consisted of flows which identified the main decision and synchronization points. Purpose of those processes is to govern development & delivery of products, and to enable different R&D centres and central functions to work together. At the low level we proposed to focus upon training, and develop supporting material

like templates, examples and checklists. The use of extensive process or procedure documents was discouraged, since they were often not used and thus brought no value for the organization. In the retrospectives that we did after every 3 week iteration we also discussed how the team member had experienced the agile way of working. It took some time initially to develop an understanding of the roles, and how to combine the team role with the local responsibilities of the team members. This happened because the team member had two roles in the team. First they represented their local R&D centre and acted as a linking pin to local process support teams, and the local management. Second they brought in their experiences and skills with process management. The aim of the team was to enable team members to contribute as much as possible from their experience to the result, increasing the effectiveness of the team as a whole. This resulted in team members picking up different tasks, depending on where they saw the biggest possible contribution. If we saw that higher priority tasks were not picked up, we discussed this in our stand-up, ensuring that they would be done in time before the iteration finished. This task approach could be conflicting with the representation role that team member had, e.g. in situation where certain process documents were to be made by a team member which would not be of use for his own organization. Where there were potential conflicts, we discussed them in our stand-up meeting, and we always managed to come to a solution. That could either be that another team member would pick up the task, or that the team member would make a first version which would then be extended by other team members based upon their experience. Though being dispersed, we managed to work together in pair where this benefited the team. All team members were very positive about the agile approach, which enabled them to contribute to the end result. Also they valued the support of the other team members, which helped them to learn from each other, and to further develop their process and change management skills.

Golden rules

The team developed and used a set of "Golden Rules". These rules helped them to better understand the agile approach, and to work together in a smooth and positive way. These golden rules were formulated based upon principles from the agile manifesto [AgileManifesto], open space technology [OpenSpace], and retrospectives [Retrospectives]. At several moments in the project, we checked if the rules were still valid and refined them to get maximum benefit out of them. Below the set of rules that we used until the end of the project:

- Dare to share–As early as possible and frequently
- The result depends on the team –Not the individual members
- The one who checks out a task is not necessarily the one who has to finish it
- The one's working on a task are the right people
- You may critique anything, but you may never criticize anyone

This simple set of rules was used throughout the project as a reminder on how we wanted to work together, they were our team values. They helped the team members to discuss and deploy the agile approach, in a very practical way. So let me go into the golden rules a little bit more.

Dare to share–As early as possible and frequently: Team members often worked in short chunks of just a couple of hours, where they produced and updated slides and documents, and reviewed the work of other team members. By sharing early we were able to continuously add value to our products, enabling delivery in short iterations.

The result depends on the team –Not the individual members: Team members frequently asked other team members to support them, or to contribute their experience or results from their own R&D centre to the project. This rule helped the team members reminding that we all brought value to the team, at different times and in different ways. Since we were all also representing our local R&D centre, this was an important value which helped the team, and the stakeholders to focus upon the contribution to the business unit result. We weren't competitors but co-workers, and the way we collaborated was beneficial for all.

The one who checks out a task is not necessarily the one who has to finish it: Team members supported each other, and collaborated where possible. It was ok for a team member to contribute just a little bit to a product, and release it for others to work on. If somebody wanted to contribute to a product that was being updated, then (s)he waited until it became available (or checked with the one working on it when it would become available), and then added his/her contribution.

The one's working on a task are the right people: We saw that when team members had the time, and the energy to work on a certain task, then they added real value to the product or service that they were working on. Team members did not wait for others to pick up tasks, but contributed when they had the possibility to do it.

You may critique anything, but you may never criticize anyone: This reminded the team to always focus on the products and services that were developed. Often it was just a matter of wording, how team members expressed their critique, but that didn't make it less important to be aware how they did it. We always assumed that people were doing the best they could, based upon what they knew and were able to do at that point in time.

These golden rules are something that many of the team members have learned from the project and are still using in their current and future work. For them it is a way to collaborate effectively and efficiently in a team.

Agile process improvement and certification

It is of course always possible to do an (class A CMMI) assessment, to get certified, also when you have taking an agile approach to process improvement. Before doing the assessment I would suggest to do some kind of gap assessment (class C or B) to identify where you are not meeting the goals of the CMMI process areas. If it is important enough for your company to be certified, then it should be easy to make a business case to implement any remaining processes or other changes that are needed to be compliant. So the agile approach to process improvement is fully possible if you want to reach a maturity level.

I actually worked with a company that was assessed to be on maturity level 3, but decided after the assessment to let their process improvement be fully driven by business needs in stead of maturity levels. They accepted that chance that they would no longer be fully level 3 compliant. My statement has always been that, if there was a need at any time to be certified again at level 3, an improvement program with a maximum lead time of 6 months should be enough to assure that all the needed processes are in place. This was based upon the capability of that organization to define and deploy processes, which could be used to introduce any process that is needed, in an efficient and quick way.

Conclusion

This paper described how we have applied our experience from developing and delivering a product in iterations (which is nowadays called agile) into process improvement projects. Scrum was used as a method to iteratively deliver products and services for a process improvement project, where a wiki and telephone conferences were the main tools to manage the work and deliver results. The agile approach has shortened the lead time in our project, gave us a better understanding of what our customer needed, and increased the commitment for the changes that were needed in the organization. The team member appreciated this way of working since it helped them to continuously contribute value, and to develop their knowledge and skills. A set of golden rules helped the team to stay aware of the way we collaborated, and work in a disciplined and effective way.

References

[AgileManifesto] Manifesto for Agile Software Development. http://agilemanifesto.org/

[Cannegieter 2008] CMMI Roadmaps. Jan Jaap Cannegieter, Andre Heijstek, Ben Linders & Rini van Solingen. CMU/SEI-2008-TN-010. November 2008.

[CMMI 2006] CMMI® for Development, Version 1.2. CMMI Product Team. CMU/SEI-2006-TR-008, August 2006

[Gilb 1988] Principles Of Software Engineering Management. Tom Gilb, Jan 1988.

[Linders 2001] Reaching Business Goals with Value Adding CMMI Assessments, Ben Linders, European SEPG conference, June 2001.

[Linders 2006] Continuous Improvement, make it visible! Ben Linders, International Conference on Software Process Improvement, April 2006.

[Linders 2009] SPI: The Agile Way! Ben Linders. Presented at the SPIder conference, October 6, 2009 in Ede, The Netherlands.
Links to be used after update of the SPIder website:
http://www.st-spider.nl/index.php?title=Conferentie_2009
Temp link: http://www.1van1.org/wiki/index.php?title=Conferentie_2009

[OpenSpace] Wikipedia Open Space, http://en.wikipedia.org/wiki/Open_Space_Technology

[Retrospectives] Retrospectives, a website dedicated to growing the practice of looking back to move forward. http://www.retrospectives.com/

[ScrumAlliance] Scrum Alliance. http://www.scrumalliance.org/

Apache ServiceMix

Axel Irriger, axel [dot] irriger [at] iteratec [dot] de
iteratec GmbH, http://www.iteratec.de

Apache ServiceMix is a runtime container for service-oriented architecture components, web services or legacy system connectivity services. It is one of the most mature, open-source implementations of an enterprise service bus and an Apache top-level project.

In this article we will take a look at what ServiceMix is and how it can be put to use.

Web Site: http://servicemix.apache.org
Version discussed: Apache ServiceMix 3.3, FUSE ESB 3.4
License & Pricing: Open Source with commercial support and packaging by FuseSource
Support: User mailing list, developer mailing list, Internet Relay Chat, FuseSource forums. Only the book “Open-Source ESBs in Action” by Manning is currently available.

Introduction

Before directly starting to discuss what Apache ServiceMix can be used for, a general understanding of the enterprise service bus acronym should be given. Since it is not a framework suited for everyday usage like Spring, the general context of its application is important.

Enterprise Service Bus (ESB) principle

Apache ServiceMix is an implementation of an enterprise service bus in the Java programming language.

In a typical enterprise environment, there are several applications installed. Most probably they are from different vendors and offer very diverse interfaces. The corporate IT commonly faces two challenges these days:

- Exchange of data between different systems
- Creation of new functionality by combining existing function block from these applications

In the past, these issues were tackled by proprietary product suites, which often were highly specialized and not able to cover a wide range of corporations' needs.

The development and adoption of web service technology is thought to solve these issues, as a common transport and description format now exists. Nonetheless, this needs to be provided, implemented and supported by the applications. Also, a single call often is not sufficient, but routing and combination logic needs to be employed, as well. To simplify these tasks, Gartner came up with the idea of an infrastructure component, which uses a harmonized way of exchanging data (messages) and provided such reusable core functionality.

This component should not only exchange messages but also incorporate the ability to route messages from one component to another component. These requirements were bundled into the term "enterprise service bus" (ESB). In contrast to enterprise application integration (EAI), the ESB concept goes beyond the mere connectivity and data exchange principles and focuses on reuse. By exposing application functionality to an ESB, new applications can be built that leverage data and services already created. These are then called “composite applications”, as

they can be thought off as a meta-level, creating applications from applications, instead of components or frameworks. This functionality can also be leveraged without using an enterprise service bus, but for the typical everyday challenges, a bus greatly simplifies things, since it is prepackaged with components and standard functionality.

Java Business Integration (JBI)

To standardize the ESB concept, the Java Community Process (JCP) came up with the JBI standard, short for Java Business Integration. It describes all components, which can be found in the ESB concept, matched to Java terminology and interfaces.

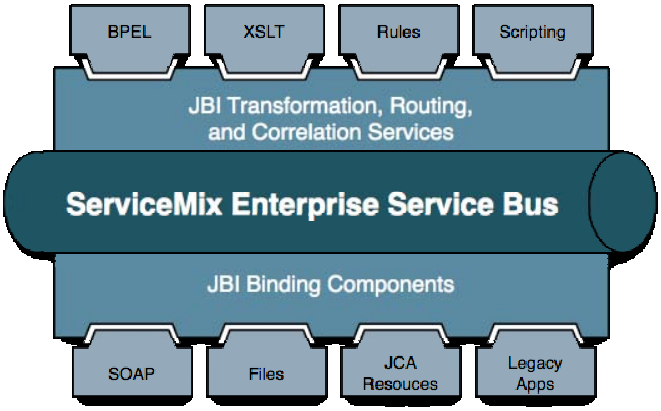


Figure 1 Overview of Apache ServiceMix (from its homepage)

Apache ServiceMix is, among other implementations, one of the most mature and implemented ESB which adhere to the JBI standard.

Components and NormalizedMessages

The JBI standard only focuses on the basic infrastructure: it defines what a component is, what a message is and how these interact. By not reinventing the wheel, a lot of terminology and concepts have been taken from the web services world, such as the supported message exchange patterns.

If you are already familiar with web services development and are not afraid of terminology like “in-only” or “in-out”, the basic ServiceMix messaging concepts will be quite familiar. Using the same terminology does not mean that ServiceMix delivers comparable functionality. In fact, it’s different.

The general idea by JBI and ServiceMix is to connect external systems to the bus by converting everything consumed externally to a NormalizedMessage. This "container" holds the actual data, along with various properties, and the destination endpoint of the message. The job of the bus is to transport the message to its destination. There, if it shall leave the bus, it is converted back to what is understood by the target system.

Within ServiceMix, all components only operate on the NormalizedMessage, which typically contains some sort of XML data, but is not restricted to it. Nonetheless, it is possible to attach binary information to the message, too.

By this, software development gets more standardized and harmonized. Once the data is delivered to the bus, it doesn't matter where it came from as it is some sort of parseable content already. This sufficiently simplifies development of third party components, which can be hooked into the bus, as these only operate on a NormalizedMessage, too.

This very basic principle, extensibility by open standards, was first introduced by Sun with the J2EE specifications, focusing on the interfaces instead of actual implementations. In spite of J2EE, a substantial number of components already exist, which can be used by ServiceMix with very little effort.

Functional overview

The ServiceMix distribution contains several components. Besides the barebone JBI bus, ServiceMix already ships with a collection of connectivity and routing components. The most commonly used are:

- Integration with JMS, which is ActiveMQ by default, but can be any JMS compliant broker
- Integration for legacy Java Beans
- Implementation of routing functionality, by means of enterprise integration patterns (EIP)
- Support for accessing and exposing services via HTTP
- Support for consuming and exposing web services via SOAP-over-HTTP or SOAP-over-JMS
- Accessing file systems
- Using rule engines, like JBoss Rules
- Using scripting languages, like Groovy

All components for external connectivity (“binding components”) come with prepackaged support for converting external data to and from XML. If this is not the desired behavior, you can customize it by implementing Java classes, which perform the conversion process. These are called marshaler and are an integral part of ServiceMix and JBI in general, too.

The principle idea of a marshaler is, that a JBI component performs a very specific task. To accomplish this, it is created and it operates only on a NormalizedMessage. To convert data to and from the NormalizedMessage, a specialized class is used, since that is no core functionality of the component and would limit its reusability.

Let's take the file binding component as an example. The core functionality is to read and write files, along with polling in intervals for new files to process. If a file is to be read, it is the function of the file binding component to lock the file, retrieve its content and send it to the bus, as a NormalizedMessage. The component does not need to know whether it’s plain XML content or something binary in the file. To actually interpret the file content, a marshaler is used.

Along with these components, you can do systems integration and provide services by wiring together existing services. You can implement some standard data interchange with it, by consuming a file, for instance and sending its content to an HTTP URL. Or you can develop something entirely different, which is then published and exposed for usage within the bus itself. But before we come to that, let's first see how ServiceMix and stuff made with it is deployed.

Deployment models

If you want ServiceMix to work for you, the question arises, on how solutions are actually deployed.

There are three scenarios supported out of the box to deploy ServiceMix:

- Embedded within your application
- Packaged in a web archive for deployment into a web container, like Tomcat
- Used stand-alone

Besides deploying and running the container itself, it is also important to get your solutions deployed, too. For this, also two deployment models exist. The "official" deployment, defined by JBI, is the "service assembly model". A service assembly is an archive, which essentially contains everything you need to run your solution. This also is the option of choice in order to remain portable across JBI containers.

The “service assembly model” is comparable to the enterprise archive (EAR) or web archive (WAR) deployment model, already in use by J(2)EE compliant servers. It contains a descriptor of the components the service assembly is made up and, for each of these, how they are configured(“service unit”).

If this is too heavy-weight for your environment or need, you can perform a lightweight deployment, which is essentially only a XML descriptor in the standard XBean notation. For simple solutions, or in an embedded use case, this is for sure the easiest way to get up and running.

Configuration

As explained above, the simplest way to deploy a ServiceMix solution is to use a XML descriptor. To get into this, we will now take a look at how ServiceMix is being configured.
Spring and XBean

The basic frameworks used by ServiceMix are Spring and XBean. Since at least Spring is well known these days, I will only cover XBean, with respect to ServiceMix, a bit.

The XBean framework was created by the Apache Geronimo project on top of Spring. The basic idea is to directly instantiate Java beans by defining XML tags. These tags are then mapped to Java beans by the framework and populated by Spring. By this, the XML configuration, typically used by Spring does not get that polluted with bean descriptors, but gets a cleaner and more readable syntax.

This whole magic works by delivering special files in the META-INF directory alongside your Java classes, which are processed at run time by the XBean platform. XBean leverages the Spring framework by providing its own ApplicationContext to enable resolution of these beans.

An example of a lightweight configuration may look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
       xmlns:sm="http://servicemix.apache.org/config/1.0"
       xmlns:foo="http://servicemix.org/demo/">

       <bean
                  id="jndi"
                  class="org.apache.xbean.spring.jndi.SpringInitialContextFactory"
                  factory-method="makeInitialContext"
                  singleton="true" />

       <sm:container id="jbi" useMBeanServer="true" createMBeanServer="true">
                  <sm:activationSpecs>
                  <sm:activationSpec>
                  ...
                  </sm:activationSpec>

                  <sm:activationSpec
                  componentName="filePoller"
                  destinationService="foo:fileSender"
                  service="foo:filePoller">
                  ...
                  </sm:activationSpec>
                  </sm:activationSpecs>
       </sm:container>
</beans>
```

In the example above you see a typical Spring configuration file. At the root element, there are namespace declarations, followed by bean definitions. If you take a look at the definition in bold letters, there is an element specified, which is within the “sm” namespace. This is a XBean definition. The XML tag “sm:container” makes the XBean framework to look for meta-data information in a package servicemix/apache/org/config/1.0 for a file, in which the element “container” is resolved to a Java class. If it fails to look this up, the whole instantiation process will fail. Otherwise (the typical case), this definition is switched to the actual bean during creation of the ApplicationContext. This bean may have properties, which get set. This happens either by using XML attributes, such as “useMBeanServer”, or by nesting elements. Either way does refer to setter-methods of the corresponding Java bean.

Within ServiceMix, both Spring and XBean are core frameworks for both the development and configuration of the platform. Therefore, you won't see any traditional bean definitions in ServiceMix, but you directly deal with somewhat “named beans”.

Although this might look a bit strange at first it actually simplifies configuration, as it makes things more readable. You can still use the traditional Spring syntax, if you like.

Lightweight mode vs. Service Assembly (SA) mode

As described earlier, ServiceMix knows two deployment scenarios for solutions: one is the standard service assembly mode, the other one is the lightweight mode.

The biggest difference is the portability issues. With both approaches, you can develop services. With the lightweight mode, you directly wire together service components in a Spring-like fashion, which results in a tightly bound application at least from the perspective of the ApplicationContext. A configuration of this style does not support hot-deployment, for instance.

The JBI way of deployment configures each service component separately as a service unit. All necessary service units are then packaged as a service assembly. This archive is deployed using the ServiceMix deployment service. In this case ServiceMix is not another bean, but used as a runtime container.

In general, if you tend to use ServiceMix in integration-like scenarios, it's easier to use it the standard way. The deployment overhead seems to be bigger, but the manageability is simplified and you are not bound to a specific product.

Example "File to JMS to file"

To get you an idea of what can be done with ServiceMix, let's take a look at a simple EAI service. We will copy a file from directory A to directory B. To stress ServiceMix a bit more, we will put a JMS queue in-between.

Although this can be also performed with other frameworks like Apache Camel, it can also be done with ServiceMix. From the general ESB point of view, this implementation scenario is only a minor one. This gets far more interesting, if you connect legacy systems to the platform and start interacting with these. But to not overcomplicate the sample here, file copy is just fine. Just make sure you keep in mind that this could also be a coupling between Siebel and SAP or some Microsoft Dynamics and a third party web service!

To implement this in a lightweight mode, you would create a servicemix.xml file, which has the overall layout like the above sample and put your component definitions into that. Upon deployment, ServiceMix will start the components in the order they are declared in this file.
To implement this as service assemblies, you need to create a service-unit for the file component, a service-unit for the JMS component and a service assembly unit, which will bundle each of the artifacts together. For examples' sake, I will not go into detail here on how to create each of these artifacts using Apache Maven, since this can be looked up easily on the ServiceMix home page, but will show you the XBean definition files for the service units.
The ServiceMix file component allows both the polling of files in directories as also the creation of files in a target directory. Therefore, we have both functionalities, which are configured in the following way:

```
<beans xmlns:file="http://servicemix.apache.org/file/1.0"
xmlns:foo="http://methodsandtools.com/samples/smx/file-jms-mover/1.0">

<file:poller service="foo:filePoller"            (1)
     endpoint="filePoller"                       (2)
     targetService="foo:JmsTarget"               (3)
     file="file:inbox"                           (4)
     autoCreateDirectory="false"/>               (5)

<file:sender service="foo:fileSender"            (6)
     endpoint="sender"                           (7)
     directory="outbox" />                       (8)

</beans>
```

This definition does the following:

(1)+(2)A file poller is created which has the service name “filePoller” and the endpoint name “filePoller” as well. If you have more than one filePoller in your service you should use the endpoint name to distinguish them.

(3) Everything processed by this component is sent to a service “JmsTarget”. Since no endpoint is specified, which is not mandatory, the first endpoint found will receive the message.

(4) The component searches a directory “inbox” for any files

(5) If this directory does not exist, it also will not be created automatically

(6) We create a target instance of the file component, which is the “sender” endpoint.

(7) It also has an endpoint name, for definition sake

(8) Everything sent to this endpoint will be written to a directory “outbox”.

This configuration file defines the basic input and output instances. Now, we need a JMS queue in between. For that, we create another XBean definition, which contains the JmsTarget endpoint. It sends messages to JMS and a JmsSource endpoint, which then reads messages from this queue, sends it to the fileSender endpoint:

```
<beans
    xmlns:jms="http://servicemix.apache.org/jms/1.0"
    xmlns:foo=" http://methodsandtools.com/samples/smx/file-jms-mover/1.0">

    <jms:consumer service="foo:JmsSource" endpoint="jms"
      targetService="foo:fileSender" targetEndpoint="sender"
      destinationName="queue/FileJmsMover" connectionFactory="#connectionFactory" />

    <jms:provider service="foo:JmsTarget" endpoint="jms"
      destinationName="queue/FileJmsMover" connectionFactory="#connectionFactory" />

    <bean id="connectionFactory"
    class="org.springframework.jndi.JndiObjectFactoryBean">
      <property name="jndiName" value="java:/XAQueueConnectionFactory"/>
    <property name="lookupOnStartup" value="true"/>
    </bean>

</beans>
```

These two definitions are all what is needed to read a file from the file system, store its content in a JMS queue and send this then back to a different directory. This is no fancy stuff, for now, but it should clarify the general idea.

Use Cases

We have seen what ServiceMix is all about and how it can be configured and used. Now, I would like to focus on typical use-case scenarios, where ServiceMix can be put to work.

Enterprise Application Integration (EAI)

Systems integration is the oldest integration issue looking for a solution. In systems integration, you have different applications, which need to exchange data, but don't use a common interchange format.

ServiceMix is useful to bridge this gap. Since it already provides access to various backend architectures and can be extended to support what is not already available, it mostly comes down to routing and data transformation. Whereas routing is a key capability of an enterprise service bus, the transformation capability results from using XSLT, Java beans or various scripting languages. Also, by using marshalers, various backend formats can be created rather easily and reused (among those, CSV and fixed-length are very typical candidates).

In contrast to other frameworks, such as Apache Camel, the ServiceMix way looks a bit cumbersome, as it feels more heavyweight. This is due to the fact that simple integration issues are not the prime scenarios for ServiceMix. It is sound practice to use ServiceMix for these types of things, although these are better put off to Camel. There, you can create your solution which is then exposed into ServiceMix as a service to be reused by other services. Since Apache Camel is shipped with ServiceMix by default, there is a tight integration already.

Service Oriented Architectures (SOA)

A service-oriented architecture is on all tables now, though it has different definitions. Nevertheless, the concept of components with defined functionality and reusability is in most definitions.

Due to the nature of ServiceMix and JBI, reusability is a key concept of the bus. The ability to integrate a wide range of services makes ServiceMix the premier runtime environment to build a SOA on. They can then be used from within the bus itself, like web services, databases or, legacy systems.

Although it should be noted that the use of an enterprise service bus in itself is in constant and open discussion, for SOA being nothing technological, you can think of components like ServiceMix as infrastructure glue, in which you embed your legacy applications. This, in itself, has nothing to do with SOA, but enables SOA as you move towards a common platform to service-ify your landscape. Also, remember that SOA does not automatically and in itself means “web services”.

Web Services

Whereas often SOA is meant, when web services are used, they don't necessarily match. Nonetheless, the development of web services is not the easiest part, due to competing frameworks and APIs. ServiceMix greatly simplifies the development, exposure and consumption of web services.

The underlying web services stack, Apache CXF, mostly reduces the development effort to annotating a Java class or creating the SOAP request via an XSLT transformation, for example. The rest is handled by the service component itself.

If you don't need the full power of a web service stack, you may also simply rely on the HTTP and JMS components. Both are able to process SOAP requests, which is enough for simple expose and consume scenarios.

This does not primarily mean that you need, or should, develop web services on ServiceMix. It is a runtime container which can expose or consume web services. Here, too, the main focus is getting data into the bus to leverage its broader functionality.

Summary

Wrap Up

We have seen in this short introduction what ServiceMix is about and what its basic architecture is. From how it can be deployed, to a sample of how it is configured, we visited some usage scenarios, like EAI and SOA.

At this point, you could see where an enterprise service bus could simplify your development efforts or daily challenges. If not, you should at least know that it is not that hard to integrate such a beast, if you need it.

Extensibility

This introduction only covered some of the components of ServiceMix. Nonetheless, you are not limited to using only these. In fact, a lot of third party components exist. They are provided by OpenESB or the directory of Open JBI components for example. If you don't find what you need there, simply develop your own components which suit your needs! As JBI is an open specification, you can continue to use them, even if you change the JBI implementation of choice, thus switching from ServiceMix to OpenESB is not that much of an issue.

Benefits

If you take a careful look at what frameworks the open source community has to offer, ServiceMix for sure is not the first choice for application development, in terms of desktop software. But if you have to deal with more complex environments, where different applications are involved and need to interact with each other, an enterprise service bus can alleviate the burden of dealing with such systems to a great extent. The concept of a standardized message model and well defined exchange patterns, alongside the benefits of components for everyday tasks simply help you focus on what is to be done and let's you not bother with the traditional "noise" in software development.

Apache Camel

Axel Irriger, axel [dot] irriger [at] iteratec [dot] de
iteratec GmbH, http://www.iteratec.de

Apache Camel is an open-source framework to exchange, route and transform data using various protocols. It is prepackaged with components for dealing with various backend systems and powerful routing and filter capabilities.

Web Site: http://camel.apache.org
Version discussed: Apache Camel 2.4.0
License & Pricing: Open Source with commercial support and packaging by FUSE
Support: User mailing list, developer mailing list, Internet Relay Chat, Fuse forums. Only the book "Camel in Action" by Manning is currently available.

1. Introduction

The need to exchange data between different applications and environments is as old as software development. Every application is built with a specific idea in mind and has its respective data model and data format most suited for the task. The task to get data exchanged among systems is always present, when dealing with more than two or three applications. Often, third party software offers export and import interfaces, such as comma-separated value (CSV), but these are most of the time not sufficient. A different data structure, the incorporation of other information pieces, or the filtering of certain parts, are typically requirements.

In the past, this was solved either by inhouse development efforts, resulting in dedicated glue or by using special, proprietary application suites. Nowadays, this can be solved with open source frameworks, bringing systems integration to a commodity level.

This article introduces you to Apache Camel, one of those integration frameworks.

1.1 Overview of Enterprise Application Integration (EAI)

During the evolution of systems and their respective integration, methodologies and patterns for typical challenges have been documented.

1.1.1 Challenges

When integrating two systems, you typically face several, quite common, issues:

- You must access and interact with the system. This is about the technical access to the system, by means of API, file access or database connectivity.
- You must transform incoming data into what is understood by the external system. This is about converting data from one format to another. This not only means data, but also file formats, protocols and alike.
- You may need to distribute data or process only specific sets of data. This deals with routing and filter capabilities.

These issues can be solved in an application specific way, although this limits reusability and slows down the process considerably by reinventing the wheel, over and over again. Therefore, best practices were extracted, discussed and documented.

1.1.2 Typical tasks

To solve connectivity issues, the typical solution or framework comes with a set of pre-packaged components for accessing often used resources, such as web services, file systems, databases, HTTP URLs.

Proprietary solutions typically also extend this to other vendor products, such as SAP or Siebel, but open source frameworks mostly limit this to open-standards resources.

Also, a typical EAI product or framework can be extended to access legacy systems, which are not covered by the vendor itself, by exposing some sort of API or software development kit (SDK).

Besides connecting to a system, data must often be transformed. These solutions cover standard based mapping technologies, such as Java beans and XSLT. In a proprietary environment, you often also find specialized tooling, such as graphical mapper frameworks.

1.1.3 Integration patterns

Since connectivity and transformation is only one side of the medal, routing and filter capabilities are essential, too. To cover those, most frameworks rely on implementing the enterprise integration patterns (EIP). These are a set of 65 patterns, which cover typical challenges.

If you apply a pattern for a problem of yours, you reduce the effort of implementation, as you use already proven knowledge, as you also increase the understanding of the problem domain for new team members. Besides that, you do not always reinvent the wheel once more.

Even situations not covered by the enterprise integration patterns directly can be solved, by combining them. If it can not be solved directly, it at least provides a proven baseline to extend from.

1.2 The Camel framework

Apache Camel is such an EAI framework, which is developed around enterprise integration patterns. It provides a decent list of pre-packaged components for accessing various backend systems and leverages a very extensible overall architecture. Since it is built with the Spring framework, it ensures a great deal of customization and extensibility.

Due to its openness, it can be deployed stand-alone as also as an embedded component within other applications or frameworks.

2. Configuration

Before digging deeper into Apache Camel itself, let's first see how it is configured.

2.1 Fluent Builder

During the design of an API, you can always choose between several approaches. In good API design, you focus on usability. Apache Camel’s focus is expressability. It evolved around the question of how easy users can combine the various functions of the API.
To simplify development, Apache Camel makes extensive use of the “fluent builder” pattern,

which focuses on expressiveness in creation. The central principle of this pattern is the definition of an interface with all the necessary methods. If a method is invoked on an object implementing this interface, all methods return this very object instance (by means of its interface). With this, the user can easily create a chain of operations on the same object, expressing logically and semantically correct, what he wants the object to do.

Consider the example of a purse. You put 15 units of money into it, then withdraw 10 and add another 5 to it. In traditional programming, you may write:

```
Purse p = new Purse();
p.add(15);
p.remove(10);
p.add(5);
```

While this may be suitable, it is difficult to understand. A more fluent example would be:

```
Purse p = new Purse();
p.putInto(15);
p.withdraw(10);
p.putInto(5);
```

This is a lot more understandable. Though, this can expressed even shorter:

```
new Purse().putInto(15).withdraw(10).putInto(5);
```

What we did here is to create a new instance of a money purse and directly work with this object to put some money into it and withdraw it. What could be done in three or four lines is put into one, making the code overall more readable and a lot easier to understand.

2.2 Notation

To configure a routing solution with Apache Camel, you have the choice of directly using Java code, which enables direct integration with any application, or by using a XBean syntax, which is the typical Spring approach.

Both approaches will be described here.

Either way, the definition consists of the same steps:

- creation of a CamelContext, which is the container for everything in Apache Camel
- definition of a data flow, which is defined as a Route. A Route always has a start point and continues with targets, to which the obtained (or created) data is sent.

Each CamelContext can hold any number of Route definitions, which enables you to logically group your data flows.

2.2.1 Spring DSL and XBean notation

To declare a route in Spring or XBean notation, all you have to do is to nest XML elements.
In Spring notation, during creation of the context, the declared beans will be created on the fly and wired together.

What you first need is a CamelContext. This is the container, which can contain any number of routes or components. Within that, you define a route.

A route has one starting point, which is described with the "from" keyword, and any number of "to" destinations, to which data is sent. Each step describes a message exchange from a source to a target destination. If, on the way, data is modified, the result will be used as input for the next exchange.

2.2.2 Java DSL

The same methodology as described for the Spring based configuration, applies for the Java DSL. Since we have some extra benefit of directly working in and with the API, the route description is briefer. Also, direct interaction with the CamelContext is a bit simplified.

3. Samples

In the previous section we have seen how Camel can be configured using both the Java API and the XBean syntax. Now, we'll see how that can be brought to action in order to actually get something done.

3.1 Example "File to file with XSLT"

The first sample is a simple one. We want to move a file from directory A to directory B. Because this is rather easy, we'll add a XSLT transformation to it.
Using the Java DSL, the definition is:

```
from("file://inbox")                (1)
     .to("xslt:someXSLT.xsl")       (2)
     .to("file://outbox");          (3)
```

What does this do?

In line (1), a route definition starts with the from keyword. As a parameter, the URL-style string defines that the file component shall be used for this. Everything after the slashes is passed as a parameter to the file component. In this case, the component scans the “inbox” folder for files and passes them on the next processing step.

In line (2), another component is instantiated. This time, the XSLT component is used with an XSL template, loaded from the Java classpath. Everything sent to this component gets transformed using that XSLT. The output of the transformation is passed on to the next step.
In line (3), another file component is used. This actually is another instance of the same component which was already configured on line (1). Everything sent to it will be written to the “outbox” folder.

As you see, this is a very straightforward and convenient way of configuring data flow. The next sample is a little bit more complex.

3.2 Example “file to database”

This sample may be used to upload data into a database.

For this to work, you must know that the JDBC component of Apache Camel sends SQL queries to a datasource. To use that component, you need to read the file content, create an SQL query from it and send this, to the database.

Using the Spring DSL, the result is (slightly modified from the Apache Camel homepage):

```
<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
     <from uri="file://inbox"/>
     <to uri="xslt:someXSL.xsl"/>
     <to uri="jdbc:testdb"/>
  </route>
</camelContext>
<!-- Just add a demo to show how to bind a date source for camel in Spring-->
<bean id="testdb"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
      <property name="url" value="jdbc:hsqldb:mem:camel_jdbc" />
      <property name="username" value="sa" />
  <property name="password" value="" />
</bean>
```

Instead of using a stylesheet to perform the transformation, it would also be valid to use a Java bean, which serves the same purpose but may perform some more powerful operations.

3.3 Example “RDBMS to JMS to file”

In the third sample, we'll take a look at how we can read data from the database and dump it to a file. Since that would be very much like what we did before, we'll add a persistence layer to it, with a JMS queue in-between.

Using the Java DSL, we get this source code:

```
from("timer://foo?period=60000").
  setBody(constant("select content from downloadTable")).
  to("jdbc:testDB").
  to("jms:queue:order ");

from("jms:queue:order").to("file://outbox");
```

The above sample does use the timer component. Every minute, the given SQL statement is executed and its result is sent to the JMS queue. From there, it is read and sent to the file system.

3.4 Extending Apache Camel

Apache Camel comes with a lot of components already. Although this is great, there probably comes the point where you need some functionality that is not already provided. For that, we must extend Camel.

3.4.1 Custom processors

The first extension is to integrate some custom processing logic within the route. For that, we add a Java processor to the routing chain. You may wonder how this is possible, but in fact, everything in Apache Camel is a sort of processor.

To realize this, you simple define a Java bean which implements the Processor interface. With this, you have full control over what you receive and what is done to the data. This Java bean is then hooked in simply by defining it as a Spring bean and passing it to the bean component, provided by Camel.

3.4.2 Custom components

Besides implementing a custom processor, which is fine for specific transformations, you can also create new components. These, you can deploy alongside Camel and use them within your routes. The dynamic lookup feature of Camel enables you to use components of which you know that they are there.

For implement a component, you create a Java bean which implements the Component interface. You then create a file in META-INF/services/org/apache/camel/component. The name of the file reflects the component name, which is used in the route definition. The content of the file simply is the full qualified name of the Java class, implementing the component. That's it!

4. Testing

After you created a solution, you surely want to test it.

One typical testing approach is to simply use live components, such as reading files from a file system, or by directly injecting data into your route.

Apache Camel supports the embedded creation of a CamelContext, which can then provide you with a client, to send messages to defined endpoints with. By this, you can send information directly from your application to Camel, thus invoking routes and possibly interacting with third party systems.

The better approach to testing instead of using real components is to use traditional concepts such as aspect oriented programming (AOP) or mock objects with Camel.

4.1 Invasive testing with Mocks

To test your solution with mock objects, you can simply change the target of a route, to the mock component. This component collects information about what is sent to it and can be queried about it afterwards. With this approach you can either test how many (or if any) messages where received by the component, or test what was actually sent to it.

If you combine these two approaches, it is possible to test whether you correctly created a route and that it works as expected.

On the contrary, using a mock needs a change of the route, which is very feasible for the development process. There, you can incrementally create your routing and see, in a step-by-step way, whether everything works as defined. When you think of automatic unit testing, this is not an option, as you don't want to change anything.

4.2 Non invasive testing with Interceptors

If you need to test stuff where you don't want to change your once defined route, the interceptor pattern can be used. An interceptor allows to transparently add functionality to an application, without changing the code. For instance, if you have a web application, you can use an

interceptor to add authentication to certain servlets. To be able to do that, it must be supported by the framework or application. Another option is to use AOP to add this, though this goes beyond this article.

Apache Camel supports the definition of interceptors before a route. These can then re-route the message flow to certain endpoints, so that you can non-invasively integrate stub implementations and therefore can extensively test your solution.
It is important to note that any interceptor must be defined prior to a route, as otherwise it would not be usable.

5. Apache Camel inside

As mentioned in the introduction, Apache Camel can be integrated with other components and can also run in an embedded fashion. Two popular projects using Camel are Apache ActiveMQ and Apache ServiceMix.

Apache ActiveMQ is a JMS provider. By using Apache Camel you get a one-stop-shopping solution for message oriented middleware (MOM) solutions. In these, ActiveMQ delivers powerful JMS capabilities whereas Camel takes the backend connectivity and routing capabilities to implement business logic. Using this combination, you can very easily decouple systems from each other with a solid JMS implementation in-between for load balancing (for instance).

Apache ServiceMix is an enterprise service bus, which actively employs Camel to implement the enterprise integration patterns. Since ServiceMix comes with a set of backend modules as well, there are various implementation alternatives. Typically, Camel is used for routing and filter capabilities and the backend connectivity is handled by ServiceMix. There are other options as well, depending on the style of usage you decided on.

The main distinction between Camel and ServiceMix is that ServiceMix aims for the larger scope of large-scale reuse in terms of connectivity, service-orientation and clustering. It therefore is more of an infrastructure on which solutions are deployed. Camel is such an solution, which can and is deployed on ServiceMix.

6. Summary

In this article, we saw how typical integration solutions can be greatly simplified by using a framework like Apache Camel. To illustrate this, several examples were given to help you get started. If you need to extend the framework beyond what is already there, you are now equipped with the basic understanding on how to create custom processor modules or even Camel components.

To complete the development cycle, we took a look at how testing is already supported by Camel, both invasively and non-invasively.

I hope you see as much value in Camel as I do, and are eager to simplify your life by testing out, how Camel can make integration easier!

ArgoUML

Franco Martinig, Martinig & Associates, http://www.martinig.ch/

ArgoUML is an open source UML modeling tool that includes support for all standard UML 1.4 diagrams. It runs on any Java platform and is available in ten languages.

Web Site: http://argouml.tigris.org/
Version Tested: ArgoUML 0.30.2, tested on Windows XP during November/December 2010
System Requirements: ArgoUML is a Java based application that needs Java 2 JRE or JDK version 1.4 or higher and 10MB of disk space
License & Pricing: Open Source**,** Eclipse Public License (EPL) 1.0.
Support: User forum: http://www.argouml-users.net/forum/

Installation

ArgoUML can be installed using the Java Web Start procedure connected to ArgoUML home page. For Windows you can also download a setup file that will install it in 30 seconds and launch the application, creating a desktop icon.

Documentation

The documentation (http://argouml.tigris.org/documentation/index.html) is impressive with different formats of a quick start and a user manual (403 pages!) that are available in English, Spanish and German. Additional ArgoUML extensions documentation and UML resources are also listed in the documentation section. An on-line tool tour allows getting a quick overview of ArgoUML interface and features.

Configuration

The configuration of ArgoUML is separated in different places. You have a classical "settings" menu where you are able to configure the user interface options like language or appearances. It is also there that you enable or disable the associated modules, like the Java code generator and some of their settings. The "Critique" menu allows to toggle design critics that create "todos" elements during your modeling activity and to adjust the importance of critics.

Features

ArgoUML support the following UML 1.4 diagram types:

- Class diagram
- Statechart diagram
- Activity diagram (including Swimlanes)
- Use Case diagram
- Collaboration diagram
- Deployment diagram (includes Object and Component diagram in one)
- Sequence diagram

ArgoUML also provides code generation for Java, C++, C#, PHP4 and PHP5. It also enables reverse engineering from Java. External modules have been developed to complement ArgoUML in specific areas. They provide generation of database schemas or code in other languages like Ruby or Delphi.

You can find a list of most of these modules on http://design.tigris.org/

In this evaluation, we will concentrate only on the UML diagramming features of ArgoUML.

User Interface

The screen of ArgoUML is split in four different panes. The "explorer" pane shows relationships between diagrams and design items according to the selected perspective. The "ToDo" pane contains the tasks that could be completed. The main window is the drawing window where you create your diagrams. On the bottom, you find a "details" pane where you can define your diagram items and link them with elements, like a "todo" item or documentation.

Drawing Diagrams

The modeling process is rather intuitive and smooth, The only missing feature is an "undo/redo" capability. The mouse gives you hints on each element that you can place on your diagram. You can then use the detail pane to describe diagram items and link to other items like documentation for instance. After each action, your model is assessed and the "todo" panel on the bottom left is updated. Clicking on one of this item will give you an explanation about its rationale and how you should act to improve your design. You can naturally turn off this constant evaluation of your design.

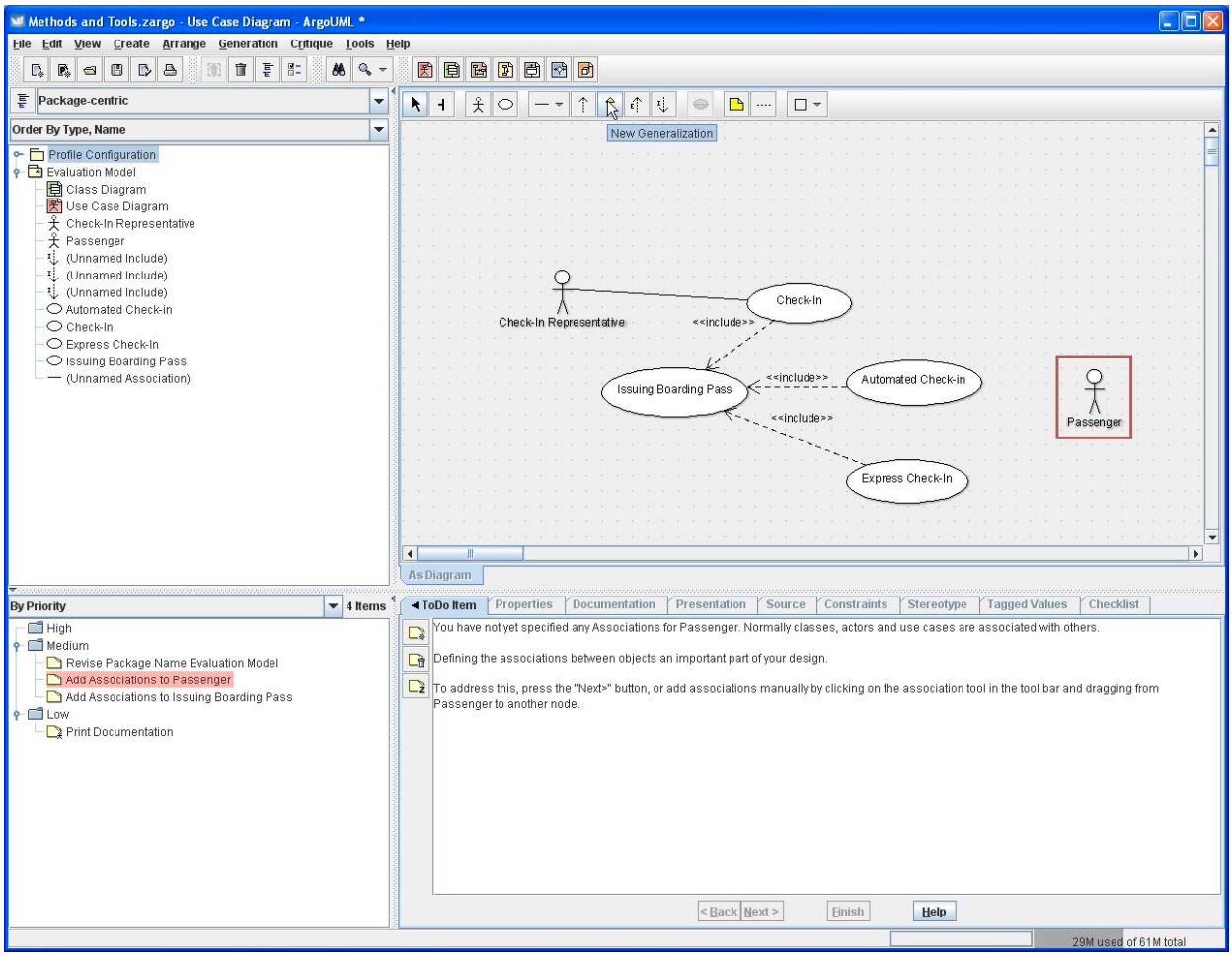


Figure 1. Design assessment during the construction of a Use Case Diagram

Another interesting feature of ArgoUML is the presence of checklist for every component of a model.

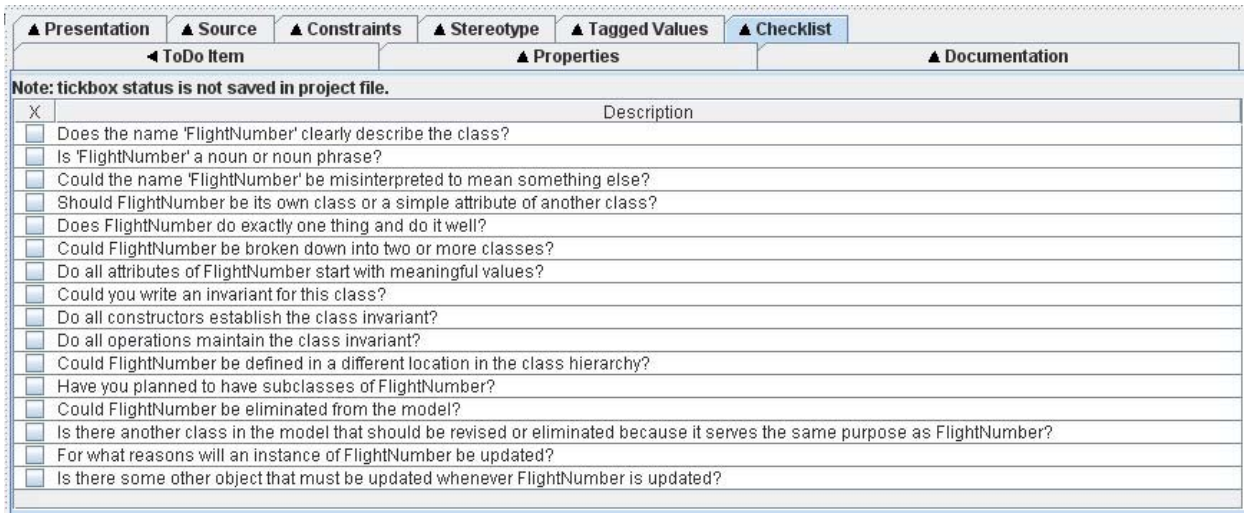


Figure 2. Checklist for a "FlightNumber" class

Class Models

Class models are influenced by generation. You can see the generated results as you are building your class and the generating process influences the design assistance. Diagrams can be graphically exported in the various formats, so that you can include them in other documents.

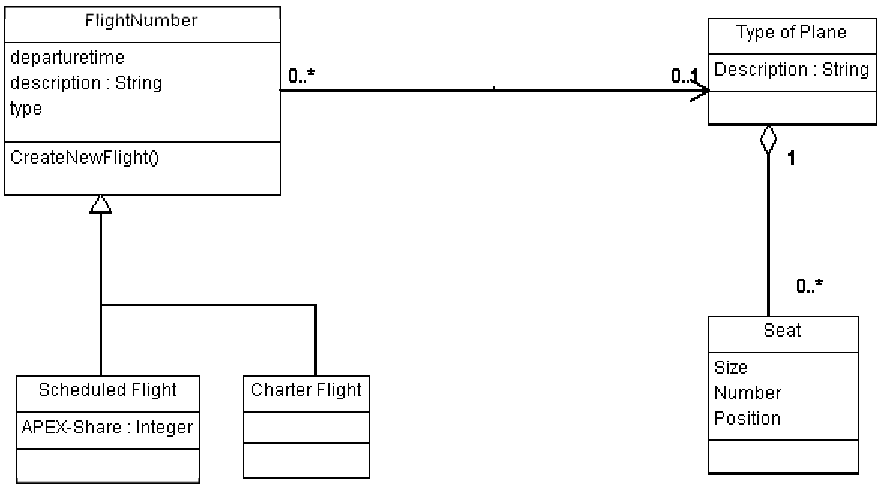


Figure 3. Exported Class Diagram

Conclusion

ArgoUML is an active open source project that provides a working tool to support basic UML modeling activities. The diagramming features are easy to use and the tools provides useful assistance in the formatting / aligning process. The nicer aspects of ArgoUML lie in its design assistance features. The design evaluation and checklists provides valuable help to make sure that your models are well formed. This will be specifically attractive for people that are learning UML diagrams or don't use them continuously. Beside the modeling aspects, ArgoUML has also some nice features like code generation and reverse engineering. I haven't checked these aspects, but if you develop with Java you might be interested to explore them.

Reference

Models of this evaluation inspired by the book "UML 2.0 in Action", Patrick Grässle, Henriette Baumann, Philippe Baumann, Packt Publishing, ISBN 1-904811-55-8

An Innovative Approach to ManagingRequirements. Integrating the requirements phase into your development process is vital to mitigating risk on large projects. This paper reviews the requirements management tools landscape and discusses MKS's unique approach, integrating requirements into its application lifecycle management platform - MKS Integrity. Learn how MKS Integrity enables you to utilize best practices such as reuse and parallel development for advanced requirements management practice. Get this Free White Paper from MKS

http://www.mks.com/mtdc-innovative-requirements-mgmt

An Introduction to Software Development. This 380 page eBook "Introduction to Software Development" is designed for IT specialists and developers that are starting their way in the free software development universe. Free Software is developed with specific collaboration techniques and tools that engage and enable world-wide communities. Professionals need to handle different programming techniques, languages and develop specific workgroup skills. This eBook also introduces information about collaborative and distributed work commonly known as "the Bazaar Model". Receive Your Complimentary eBook Now!

http://methodsandtools.tradepub.com/free/w_free03/

Advertising for a new Web development tool? Looking to recruit software developers? Promoting a conference or a book? Organizing software development training? This classified section is waiting for you at the price of US $ 30 each line. Reach more than 50'000 web-savvy software developers and project managers worldwide with a classified advertisement in Methods & Tools. Without counting the 1000s that download the issue each month without being registered and the 60'000 visitors/month of our web sites! To advertise in this section or to place a page ad simply http://www.methodsandtools.com/advertise.php

METHODS & TOOLS is published by **Martinig & Associates,** Rue des Marronniers 25,
CH-1800 Vevey, Switzerland Tel. +41 21 922 13 00 Fax +41 21 921 23 53 www.martinig.ch
Editor: Franco Martinig ISSN 1661-402X
Free subscription on : http://www.methodsandtools.com/forms/submt.php