Issues in Scaling Agile Using an Architecture-Centric Approach: A Tool-Based Solution

Kris Read, Frank Maurer

University of Calgary, Department of Computer Science
{readk, maurer}@cpsc.ucalgary.ca

Abstract. Agile software development processes are best applied to small teams on small to medium sized projects. Scaling agile methodologies is desired in order to bring the benefits of agile to larger, more complex projects. One way to scale agile methods is via an architecture-centric approach, in which a project is divided into smaller modules on which sub teams can use agile effectively. However, a problem with architecture-centric modifications to agile methods is the introduction of non-agile elements, for instance up-front design and integration difficulties. These issues are discussed and a tool-based solution is presented facilitating the adoption of the architecture-centric agile approach.

Keywords. Agile Methods, Scaling, CruiseControl, Continuous Integration, Test Driven Design, Automated Testing

1 Introduction

Martin Fowler likes to say, "Scaling agile methods is the last thing you want to do.[1]" At the Canadian Workshop on Scaling Agile Processes this generated quite a stir, but it turns out that he meant it literally. The idea is that one should examine every other alternative first, and consider scaling as a last resort. Nonetheless there is a need to scale agile methods. Large projects are out there, projects for which a small team is not ideally suited. If a team needs to deliver a lot of functionality but also has a lot of time, the team size can be quite small. Likewise the team can be small if it has not much time but can reduce the scope of the project. However, to deliver a lot of functionality in a short amount of time, the business solution is to add more people. Scaling a software development project would traditionally be accomplished through heavyweight processes and stacks of documentation. But it is desirable to reduce the project overhead in order to maximize productivity, and so the question becomes "How do we scale Agile Methods?" To improve the scalability of agile software processes, one solution is to follow a divide and conquer strategy based on architecture.

An architecture-centric strategy is nothing new – Ken Schwaber advocates using the first iteration of an agile project to have a smaller team define the project architecture, and then proposes multi-team coordination through a "Scrum of Scrums" for the remainder of iterations. If the project is initially broken down into smaller modules, each module can be built using an agile approach. This plan enables the application of proven agile methodologies using small cohesive teams at a module level. Following this strategy may also enable distributed software development in an

[1] Keynote address, Canadian Workshop on Scaling Agile Methods, 2003. http://can.cpsc.ucalgary.ca/ws2003

agile way. Agile depends upon co-located teams for close communication, but if a project were properly divided each sub-team could independently follow an agile process. In addition, if organizations are interested in exploiting the commonalities between its products or systems, an architecture-centric strategy may improve code re-use through the definition of modules. However, there is in fact an intrinsic contradiction between agile software development and the practice of separating a project into modules. By adopting such a strategy, will our process remain agile? Common sense says that there will be several incompatibilities between agile processes and the architecture-centric approach. These incompatibilities include up-front design, team inter-communication and module integration. This paper proposes that these problems of architecture-centric agile software development can be overcome through innovative tool support.

2 Concept

We sometimes assume that a comprehensive document is necessary for architecture-centric development, or that every team needs to know precisely how their product depends upon products developed elsewhere in order to construct it. This approach, however, is the antithesis of the maxim “Responding to Change over Following a Plan” stated in the Agile Manifesto[2]. Up-front planning can still be done in an agile way, so long as we stay focused on doing only what is required. In fact, agile projects normally have some overhead when user stories are gathered and prioritized, development tools chosen, environments configured, and so forth. Defining the system architecture can be included as one of the aforementioned startup costs, if the architecture is defined in a quick, lightweight manner that is flexible to change. The best way to assist agile developers with quickly generating such a definition is to provide a simple tool that they themselves can understand and work with.

In an ideal world, modules would work flawlessly with one another, and there would be no integration problems. Anyone who has tried integration knows that this is rarely the case. The interfaces between modules are problematic; even if these interfaces are well documented, it is possible that over time requirements changes or lack of communication between parties will result in incompatibilities. Without knowledge of exactly how outputs are going to be used, there is no guarantee that developers will be able to deliver them as expected. To address this issue, one can apply the same concept of continuous integration already utilized by agile teams.

> “An important part of any software development process is getting reliable builds of the software. Despite it's importance, we are often surprised when this isn't done. We stress a fully automated and reproducible build, including testing, that runs many times a day. This allows each developer to integrate daily thus reducing integration problems.” [3]

In an architecture-centric agile environment, it is not enough to simply perform an automated build and test whenever there is a change to the system. Because each module is assuming that its fellow modules will be constructed according to the

[2] Agile Alliance, Manifesto Website
http://www.agilemanifesto.org
[3] *http://cruisecontrol.sourceforge.net*

architecture, tests based on the same (possibly incorrect) assumptions do not indicate the health of the system. When Jack is developing a module, it does the project little good if Jack also writes tests for his interface. Jack may be very well aware of what functionality he is providing, but likely has no knowledge of the functionality that other modules are expecting him to provide. It is thus very probable that Jill's module, which uses the module written by Jack, will have some specific need Jack knows nothing about. Conventional continuous integration should still be done for each module, but there must also be a higher level of continuous integration to ensure compatibility between modules even before they are implemented. It is therefore desirable to extend the concept of continuous integration such that some kind of quality assurance and verification of the interfaces is performed automatically with each build. The key to this continuous integration at the module level is getting the tests right.

The most effective arrangement would be for Jill to act as a customer for Jack at the module level. Jill will write tests for the functionality that she expects from Jack, and for her to do this *before* Jack writes his actual code. Jill doesn't need to test Jack's entire interface, just the features that she herself will be using. Thinking of testing before doing the development is not exclusive to agile; the "V-model" adaptation of waterfall[4] is one of the simplest examples of this, when you plan ahead to use your design documents and specification documents to verify your product. Agile processes can replace the "V-model" comparison of functional specifications to code with automated unit tests; this new concept can replace comparing an architecture specification to developed interfaces. The idea of API consumers writing tests is similar to that already discussed by Newkirk[5] for doing test first design of third party software. Newkirk asserts that in addition to writing tests before writing code, you should write tests before *using* code written by others. However in this case, the third party software itself may not have been written yet. You are tailoring the tests as much to your own requirements as to the functionality that will finally be provided.

This extension of "test first" design could have quite a few benefits. Any problems in the existing architecture would be uncovered early on in the iteration by test authors. Incompatible tests or conflicting tests will reveal problems in the architecture before more effort is wasted. Following this plan would enable an evolution of the system architecture; just as doing test first design for regular code helps you think and plan ahead better, so will doing test first design for module interfaces let you look ahead and construct your architecture. This evolution of the architecture should also involve actual customer representatives, who can make decisions about the entire deliverable system if conflicts or questions of priority arise. If a change in requirements influences the system architecture, new tests that verify the new functionality or structure can be added. The continuous integration software can facilitate this by notifying affected teams when changes are made. Changing the architecture drastically is potentially a source of difficulty, but to address this we can recall how refactoring handles changes to code. Changing a small amount of code can sometimes have sweeping effects, but now and again we need to evaluate the cost-benefit tradeoff and make a decision. If our general strategy is to make changes little by little, and keep the architecture healthy, then flexibility is not necessarily lost. Teams will not only have access to the architecture definition describing the modules,

[4] Daich, G: Software Test Technologies Report. 1994
[5] Newkirk, J.: A Light in a Dark Place: Test-driven Development of 3rd Party Packages. 2002

but also to a set of tests representing the functionality that they need to implement; they can use these tests both as a knowledge sharing mechanism and as a contract between modules. Jack knows he is finished when all of Jill's tests pass. Likewise, Jill knows Jack's code will integrate with her own when the tests she provided are successfully run by Jack. In essence, the author of a test becomes a customer for the module developer. A hierarchy of customers is formed, with one or more actual on-site customers at the top. The real customers speak with some of the teams, who then define user stories and tests related to the child components. At each level the developers have their own product backlog of user stories defined by the customers with whom they interact. These user stories are complimented by the automated tests.

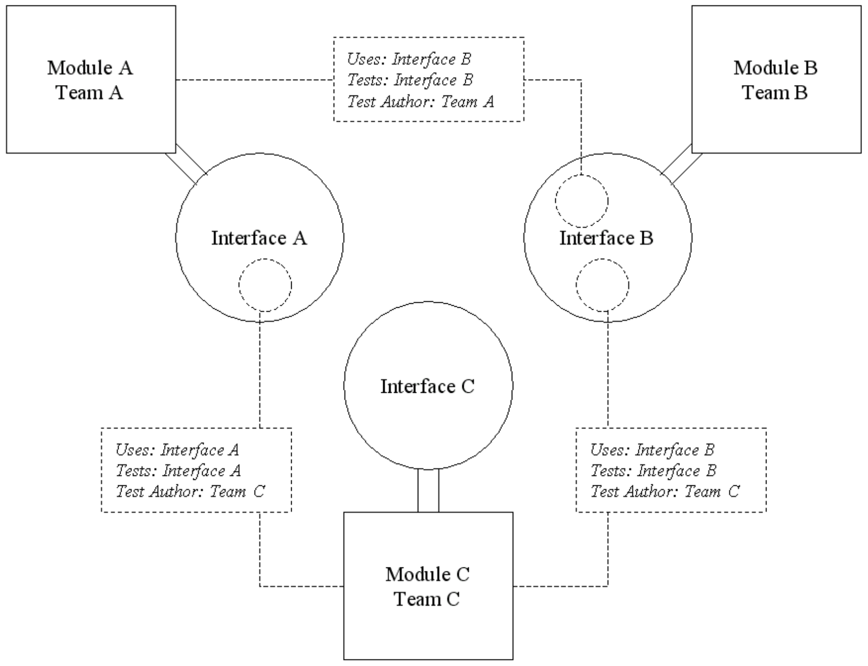


Fig. 1. Each team will do test first design for the portions of other modules' interfaces that they use. In this example, two modules depend on Interface B and one depends on Interface A. This means that three test suites will be written before development

In summary, architecture-centric software development can be combined with agile software development processes while retaining the spirit of agile by following these guidelines:

1. Design the module architecture in a quick and lightweight way
2. Provide the architecture in a format that is flexible to change
3. Require test first design at the interface level
4. Tests are written by module users not module providers
5. Test authors act as customers for dependant modules
   (in addition to real on-site customers)

6. Module teams define their own product backlog of user stories
7. Do continuous integration of the module based system

3 The Tool: COACH-IT

At the University of Calgary work is being done on a lightweight architecture planning and continuous integration tool for agile processes. **COACH-IT**, the **C**omponent **O**riented **A**gile **C**ollaborative **H**andler of **I**ntegration and **T**esting, is an effort to develop tool support for scaling agile practices using an architecture-centric approach. The sequence executed by COACH-IT is as follows:

1. Users define an architecture using the COACH-IT input web application
2. Multiple repositories are monitored for code changes in each module
3. When a change is detected the module and related modules are downloaded
4. The modules are deployed and tests are run to ensure interface compatibility
5. Teams are notified directly of any problems via electronic mail
6. The "health" of the system is available to the teams via a web page

COACH-IT combines and extends existing continuous integration technologies in order to provide an end-to-end solution for module definition and testing. The following diagram shows the interaction of COACH-IT technologies:

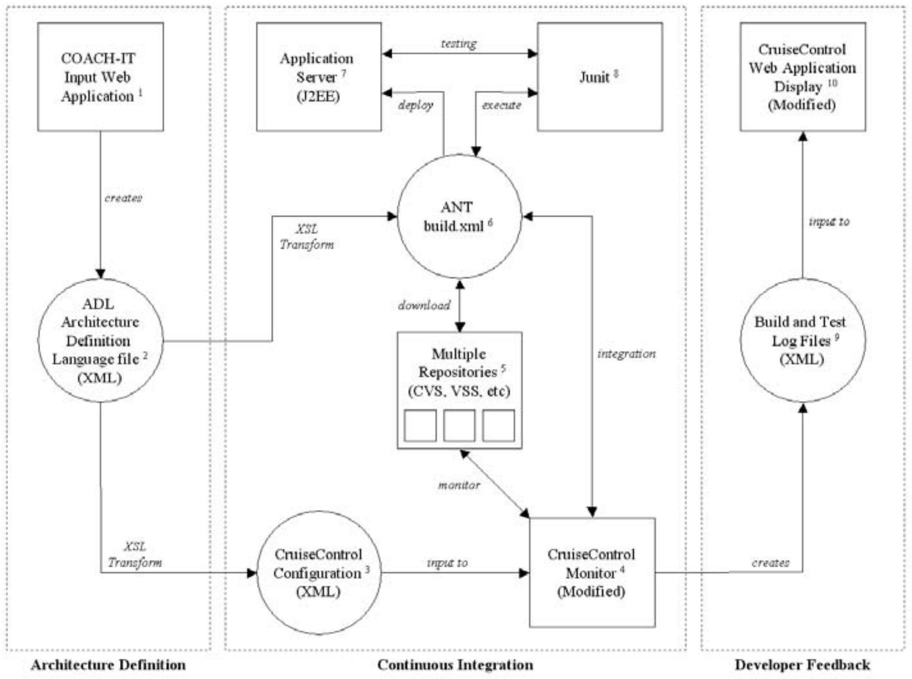

COACH-IT Input Web Application [1]
creates
ADL Architecture Definition Language file [2] (XML)
XSL Transform
XSL Transform
Application Server [7] (J2EE)
testing
Junit [8]
deploy
execute
ANT build.xml [6]
download
Multiple Repositories [5] (CVS, VSS, etc)
integration
monitor
CruiseControl Configuration [3] (XML)
input to
CruiseControl Monitor [4] (Modified)
creates
Build and Test Log Files [9] (XML)
input to
CruiseControl Web Application Display [10] (Modified)
Architecture Definition
Continuous Integration
Developer Feedback

Fig. 2. Above is a conceptual drawing of how COACH-IT works. The tool has three main functions: Architecture Definition, Continuous Integration and Developer Feedback. Note: superscript references in Section 3 refer to entities in Fig. 2

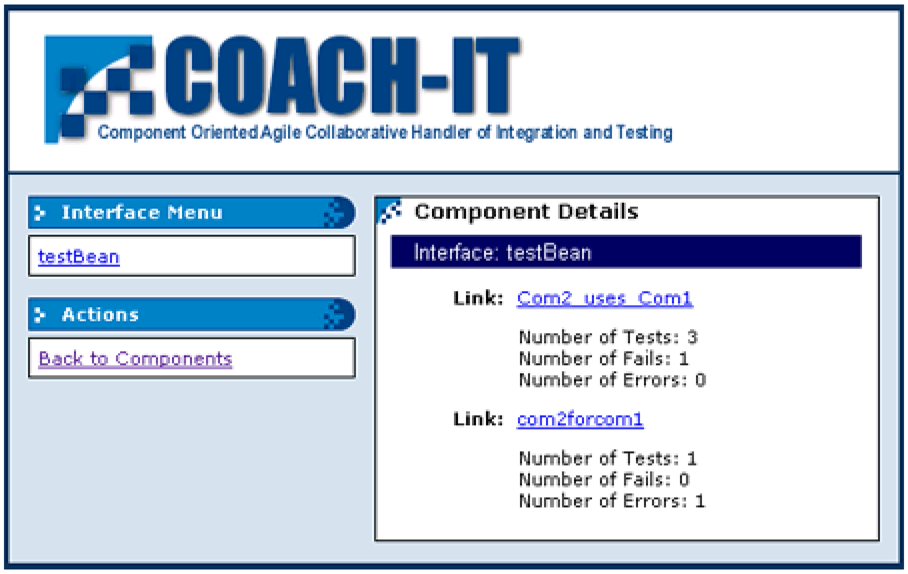
The COACH-IT Input Web Application has been designed to assist agile practitioners with managing architecture definitions. In an agile project the focus is on producing value for the customer, and the architecture definition itself is not a deliverable. Using the COACH-IT tool any developer can quickly define a set of modules and assign JUnit tests to the interfaces between those modules, thus minimizing design overhead. A web application[11] provides a simple to use, self-documenting interface with which most developers are already familiar. The same application can also load and edit current or previous architecture definitions; architecture definitions in an agile project are likely to change. Although even the minimum necessary ADL can become complex, a web interface hides this complexity and lets the developer concentrate on delivering something real.

The core of COACH-IT is the Architecture Definition Language file (ADL file)[2] . This file is a minimalist representation of the modules, interfaces and relationships in the system. Defined within this file are module names and (optionally) descriptions/annotations, module repository locations, module file locations, module interfaces, module team contact information (e-mail), module relationships (unidirectional), relationship test associations, test repository locations, test file locations and test contact information (e-mail). Only these few items are required as user input to create a simple architecture for continuous integration. The ADL file is stored as XML, which makes it both extensible and flexible. Moreover, XML is easily formatted for human viewing and is familiar to many developers. Finally, COACH-IT uses XML as its document format so that it can be integrated with existing and future tools that use XML as input and output. The core technologies underlying COACH-IT (ANT and CruiseControl) both rely heavily on XML, and therefore using XSL to generate required files makes sense. A sample COACH-IT ADL and the latest schema are available on the COACH-IT home page, but are not included here.

COACH-IT determines when modules are changed using a modified version of the CruiseControl continuous integration tool[4]. The primary modification made to CruiseControl allows the monitoring of multiple repositories, which are then monitored individually according to custom settings and schedules. Each team is thus able to configure their own repository to suit their unique needs[5]. Input to the CruiseControl monitor is via an XML file generated from the ADL using an XSL script[3]. The CruiseControl configuration file follows the standard CruiseControl format but allows multiple project definitions (one for each module). More information on CruiseControl is available at (http://crusecontrol.sourceforge.net). When COACH-IT detects a changed module it calls an ANT build file to perform the integration and testing[6]. This ANT file is likewise generated via the ADL file using XSL. Each component will have one ANT file that will download the module and any other dependant modules, deploy them on the application server[7] and run the suite(s) of associated JUnit tests[8]. Because these ANT files are generated using XSL scripts it is simple to add additional ANT tasks if required; for more information about ANT visit the Apache ANT page at (http://ant.apache.org).

COACH-IT is also able to directly notify teams and individual developers via electronic mail. In the event of a test failure or other change in system health, COACH-IT can be configured to notify any and all involved parties, such as the

[1 … 10] References to entities in Figure 2 authors responsible for the test, the authors of the involved modules, the developers who last committed, the team leaders, or the entire teams of the failed components. This direct notification is a key component to why continuous integration is effective. Alistair Cockburn has defined the concept of "information radiators" as anything that will "increase team communication without unnecessary disruption" (Cockburn, 2003). The goal of COACH-IT is partly to act as such a radiator, providing as much information as possible through everyday channels.


COACH-IT
Component Oriented Agile Collaborative Handler of Integration and Testing
Interface Menu
testBean
Actions
Back to Components
Component Details

Interface: testBean

Link: Com2 uses Com1

Number of Tests: 3
Number of Fails: 1
Number of Errors: 0

Link: com2forcom1

Number of Tests: 1
Number of Fails: 0
Number of Errors: 1

Fig. 3. Health of the system can be viewed for components, interfaces and relationships. Initially a brief summary is shown, more detail is available by clicking on the links

Output from the CruiseControl monitor is also in standard CruiseControl XML format[9]. In fact, each module creates its own logs compatible with the standard CruiseControl web application. However, COACH-IT also includes a custom web application based partially on CruiseControl that summarizes the results of tests across the entire architecture[10]. Details and contact information are provided for each test in the event of a failure. There is also a history feature that allows the user to browse through past tests and system states interactively.

5 State of Implementation

COACH-IT is being developed using JAVA, XSL and XML technologies, builds on CruiseControl and Apache ANT, and runs on a free, open-source platform. At the present time COACH-IT is able to monitor multiple J2EE components in multiple repositories, downloading, deploying and testing them as required. Our ADL file definition is stable and can be verified against an XML schema. Furthermore, the COACH-IT web interface allows simple interactive editing and creation of ADL files as well as an overall display of system health. COACH-IT is at the stage where it can be self-hosted. In fact, COACH-IT has been designed in a modular way and is therefore quite suitable for development using the previously discussed approach. If you would like to see a demo of the system, or download it for your own use, please contact the authors.

6 Future Work

Future work on COACH-IT first includes further refinements to the output web-application with the goal of constantly giving teams as much information as possible about their own component as well as the entire system. The COACH-IT system also needs to be generalized in such a way as to be applicable to non-J2EE projects. Conceptually, COACH-IT can easily be integrated with existing visual modeling (UML) tools through our XML based architecture definition. Conversion allowing users of popular industry modeling tools to directly import their component structures into COACH-IT is on the horizon. We would also like to integrate COACH-IT with MASE, a tool to support agile planning and estimation developed at the University of Calgary. MASE will facilitate developer and team communication in a non-intrusive manner.

A study of projects developed using an architecture-centric agile process with tool support is in the planning stages. This study will be collecting data to evaluate the productivity and/or satisfaction of teams using COACH-IT under the described methodology. In the future COACH-IT should also be compared with other tools used to keep track of the state of a system under development, and possibly incorporate some of the compatible features of these systems.

7 Conclusion and Potential Problems

The architecture-centric strategy is still open to some criticism. Yes, there will be some overhead in maintaining the architecture definition, even if this overhead is lessened through tool support. However, there is always a minimal amount of documentation necessary to help the developers do their work. To quote to Kent Beck, “Contrary to the claims of some of XP's detractors you do in fact invest time modeling when taking an XP approach, but only when you have no other choice. Sometimes it is significantly more productive for a developer to draw some bubbles and lines … than it is simply start hacking out code” (Beck, 2000). This approach was also designed with an object-oriented refactoring environment in mind, and so may not be applicable to other project types. Moreover, A team management process, like Scrum, is essential when working on a large or distributed agile project. COACH-IT, and the concepts proposed above, are meant to compliment existing agile processes. Lastly, there is an element of trust involved, as in many agile practices. COACH-IT does not restrict individuals from changing the architecture or tests at whim. Although this attitude may work well for some teams, there is no solid data to defend it yet. The concept and tool will undoubtedly be improved with experience, but by combining lightweight planning with an architecture-centric design strategy we hope to get the most benefit without compromising the spirit or practices of agile methods.

8 Acknowledgements

The COACH-IT Software is based heavily upon the CruiseControl Continuous Integration Toolkit and Apache Ant. Credit should also go to xADL, an XML-based ADL developed by the Institute for Software Research at the University of California, Irvine. The COACH-IT ADL is based roughly on the concepts and methods of xADL.

Work on implementing COACH-IT has involved the efforts of graduate students in a Distributed Software Engineering course (CPSC601.85) at the University of Calgary and the University of Alberta. In alphabetical order: Yichuan Cao, Amy Law, Tracy Li, Zhizhong Li, Anny Lin, Bill Luthi, Kris Read, Lance Titchkosky, Eileen Wang, and Fakui Wang.

References

1. Agile Alliance Home Page. Web, 2003. http://www.agilealliance.com
2. Ambler, S.: Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. John Wiley & Sons, February, 2002
3. Beck, K.: eXtreme Programming Explained. Addison Wesley, 2000
4. Canadian Invited Workshop on Scaling XP/Agile Methods. Proceedings, 2003. http://can.cpsc.ucalgary.ca/ws2003/
5. COACH-IT Home Page. Web, 2003 http://pages.cpsc.ucalgary.ca/~readk/COACH-IT
6. Cockburn, A.: Crystal Clear: A Human-Powered Methodology for Small Teams. Draft, 2003. http://members.aol.com/acockburn/
7. CruiseControl Home Page. Web, 2003. http://cruisecontrol.sourceforge.net
8. Fowler, M.: Continuous Integration. Web, 2003 http://www.martinfowler.com/articles/continuousIntegration.html
9. Daich, G., Price, G., Ragland, B., Dawood, M.: Software Test Technologies Report. Software Technology Support Center, Hill Air Force Base, Utah. 1994
10. Newkirk, J.: A Light in a Dark Place: Test-driven Development of 3rd Party Packages. XP Agile Universe, 2002
11. Schwaber, K.: Agile Software Development with SCRUM. Prentice Hall, 2001
12. xADL Home Page. Web, 2003. http://www.isr.uci.edu/projects/xarchuci/