

MAXIMIZING PRODUCT
DEVELOPMENT ROI™SPECIAL
POINTS OF
INTEREST:

- Lean gives us three kinds of TDD
- Implementing Scrum for Your Team —the effective way to get started
- Why does Scrum Work
- Velocity is the measure you want
- Coaching challenges

INSIDE
THIS ISSUE:

Challenging 2
why (not if)
Scrum
works

Velocity is 3
the meas-
ure

Completing 4
the Agile
Develop-
ment Puz-
zle

Coaches' 5
Corner

Upcoming 6
courses and
seminars

Net Objectives EZine

VOLUME 4, NUMBER 1

JUNE 2007

The Ascension of Test-Driven Development

Test-Driven Development is gaining traction in the community. At Net Objectives, we are very happy to see this. I remember getting into trouble at times years ago because I kept calling it Test-Driven *Design* instead of Test-Driven *Development*. My rationale was that when you considered how you were going to test code, you were also designing it. Have you ever said to yourself, "I wish I (they) had considered how I was going to test this code before I (they) wrote it"? If so, you know what I mean. Considering how to test code relates strongly to issues of coupling, cohesion, redundancy, readability, focus and encapsulation. These are pretty much all of the code qualities we consider important (the

seventh is testability; for a discussion, see the Code Qualities [streamzine](http://www.netobjectives.com/resources/streamzines) at www.netobjectives.com/resources/streamzines. Note that you will have to register first before you can see it.)

I am particularly excited to see that the *scope* of Test-Driven Development is expanding.

Originally envisioned as a process that an individual or a pair of programmers would use to create code, I suggest that TDD can now encompass the entire development team, including the customer, product owner, business analyst, tester and QA roles.

While it is all TDD, it is helpful to create some vocabulary, to make distinctions about the various ways TDD is addressing the

needs of software development in a wider way. This article proposes three such distinctions.

Functional Test-Driven Development

To begin, let's look at how TDD is being used at different levels in an organization.

Start with classic TDD, which I will now call *Functional Test-Driven-Development* (FTDD). Functional TDD uses tests to assist in the creation of specified functionality. A developer (or pair of developers, and so on throughout this article) gets a requirement that she needs to implement. Often, this requirement comes in the form of a story. Instead of figuring out how to implement the story (design), code it (implementation), and then test it, the developer follows the technique (which may feel non-intuitive at first) of specifying the test first. I would

(Continued on page 2)

Announcing a New Course: Implementing Scrum for Your Team

Too often, Scrum training leaves teams with a vague feeling that they do not know what to do next. What are the practical steps to take? How do they adapt the principles they have just learned to the realities of their own situation? What is essential to do right now?

To address this, Net Objectives

has created a new series of classes called *Implementing Scrum for Your Team*.

This is a course for your entire team, ideally something to take right at the start of a project. While only one or two team-members need to know how to

(Continued on page 4)



Alan Shalloway is CEO and Founder of Net Objectives. He is a Certified Scrum Master

and Author of *Design Patterns Explained*. With a MS in Computer Science from M.I.T., Alan's passion is to help create environments where software can be developed effectively, without suffering.

blog

blogs.netobjectives.com

CHALLENGING WHY (NOT IF) SCRUM WORKS

by Alan Shalloway
May 25, 2007

I have repeatedly heard that “Scrum succeeds largely because the people doing the work define how to do the work.” That is even a *third* of why Scrum works.

In the last 7 years, as a trainer/coach in Lean and Agile methods, I have had the privilege of seeing dozens of teams migrate to Lean-Agile methods. As I work with these teams, I am always watching to see what helps to shift people’s behavior (not just their understanding).

Shifting behavior requires more than simply knowing what to do and how to do it. It also requires more than having done it a few times. It requires a confidence that what they will do will make a difference when things get tough because, when things get tough, people tend to go back to their previous (bad) habits. They begin to fear that they will not succeed with the new approach, so they revert to what they have been doing for years, even if they’ve had short-term success with better methods.

It’s a good thing that we didn’t have our attitude about learning as babies that we have now. Otherwise, we’d never have learned how to walk!

So, why does Scrum work?

read the post at
blogs.netobjectives.com

The Ascension of TDD (cont)

(Continued from page 1)

claim that this still constitutes analysis and design, but in the form of a test. In other words, the developer is asking “How can I test for such and such a function” where the function specified is some subset of the entire function being requested. She starts with core functionality. Essentially, she has analyzed the situation and decided where to start.

The next step is asking, “and how will I know when I have done this?” The answer is a specification of inputs and outputs that can be validated. This specification accomplishes two things:

- It provides a test that can be run against the software
- It provides a black-box design of a component of the system

By dealing with only core functionality at the start, it is likely the developer’s solution will have strong cohesion (since it is focused on one thing), little coupling (since the developer is motivated to keep the test simple and straightforward) and no redundancy. If the test specification indicates that weak cohesion or tight coupling is present, the developer should refactor her code to eliminate that need. This illustrates how test specification is the first step towards maintaining a quality design.

The next step is to implement the test from the outside in; that is, to call the function which should fail (since the code is not implemented yet). This validates

the test (if it succeeds then either the developer’s understanding is incorrect or the test is useless). Once the test has been turned into this kind of “executable specification” (as we like to call them) the code can be implemented. We repeat this process until the required functionality has been specified.

Note that this process of implementing the test first has accomplished the following:

- Analyzed the function to determine where to start its implementation
- Defined the test
- Considered the impact the test will have on the code
- Refactored the code to keep its quality high
- Implemented the test
- Repeated until the functionality is complete

I call this *Functional Test-Driven-Development* because it is using TDD to create functionality.

Acceptance Test-Driven Development

The process of generating automated acceptance tests is not exactly the same as, nor does it serve exactly the same purpose as automated functional testing. Since it is not the same, it seems best to give it new label; so, I will call the process of generating acceptance tests prior to writing code and then implementing the

code to get the acceptance tests running *Acceptance Test-Driven Development (ATDD)*.

In Acceptance TDD, the developers must talk with both quality assurance and the business analysts to determine what the requirement (again, often in the form of a story) means. Again, the question – “and how will I know I’ve done that” is still a good one. Only now, the developers are talking to others, not themselves.

The specified test is a specification for the story. If it is implemented as a test it becomes an executable specification. Note that specifying it at the beginning has adds no extra work and can

(Continued on page 3)

Do you suffer from RDD?

Tom Poppendieck describes the phenomenon of teams creating more code than acceptance tests. For most teams, the gap between the functionality they have and the automated tests for them grows over time. This deficit causes severe problems during regression testing. Hence, the term Regression Deficit Disorder or RDD.

Agile teams are discovering that the best, perhaps only, way to address RDD is to use automated testing. This involves not only automated functional testing but also automated regression testing. The involves migrating from automated unit-testing (using something like XUnit) to an automated acceptance testing tool, such as Framework for Integrated Tests (FIT) and Finesse. In the crush of development schedules, automating all aspects of testing is the only way to avoid RDD.

The Ascension of TDD (cont)

(Continued from page 2)

assist in the communication of the story. As Rick Mugridge describes in his seminal text *FIT for Developing Software* (a fabulous book on the FIT tool and on what I am calling Acceptance TDD), specifying tests up-front changes the nature of the communication from a conceptual conversation to a blend of concept (the story) and implementation (the test). Conversations of this type are much better than conversations that deal only with concepts or implementations. This is because the implementation aspect provides a validity check for the conceptual aspect of the conversation.

If a testing tool such as FIT is used, Acceptance TDD requires

Developers must talk with QA and the Business. They all have a stake in this testing business

testing methods use. Note, in both cases we have to specify the test. Using FIT, we specify it in the tool. All that needs to happen is to tie the tests to the tool (i.e., write the “fixtures” as they are called). This involves very little extra effort and offers us many benefits, such as the following:

- Specifying tests up-front dramatically improves communication.

- Developers are more likely to understand what is required
- Customers sometimes see that what they are asking for is not what they actually want
- Developers have a target to shoot for
- Automating tests gives the developer an ongoing indicator as to whether code is working or broken.
- The suite of tests provides automatic regression tests to cover future maintenance

Acceptance TDD is really a team approach, involving the entire team: developers, QA, Business Analysts, testers. And increasing collaboration is a good thing.

Lean Test-Driven Development

There is still another type of TDD, which I call *Lean Test-Driven Development* (LTDD). Lean is the generic term used for Toyota’s philosophy of providing people with a process for assisting their work. When it comes to TDD, the lean principles that are most important: seeing process as a baseline; eschewing workarounds; and embracing the discipline of QA.

Lean says that processes exist to assist the team; they are not something the team does just because it is told to. The process is a baseline for change – it provides us with the best way of writing software that we know of. Certainly, the process will

change because the team is always learning more and because it has to adapt to changing environments, requirements, etc. Lean thinking has a bedrock belief in local knowledge, respecting people and what they know to improve what they do and then committing together to do what has been agreed to be done. Thus, we commit to following the process until we know it needs to be changed; then we change it, and continue. Change is inevitable and is something to be embraced.

Another key principle in Lean is that we do not tolerate workarounds. If we have a problem, we fix it right away. This ties in nicely with yet another principle – impediments to delivering value (flow) is considered waste. Creating bugs and implementing the wrong requirements (those that don’t provide value – for whatever reason) are clearly impediments to adding value.

Lean tells us to use the quality assurance process not to just validate that the product (our software) is of high enough quality (i.e., it does not have bugs) but also that it can assist us in improving the process we are following to generate our product (code). This view comes from the use of process in Lean.

Lean concentrates on building a great process because it has been demonstrated that most errors are due to the process being followed. Hence, in Lean, defects are considered the failure of the process, not the failure of the people following the

(Continued on page 4)

IN AGILE, VELOCITY IS THE MEASURE YOU WANT

by Bob Hartman
May 15, 2007

When a development team starts using an agile process one of the first questions asked is: “What is it important for us to know?” I believe there are many agile consultants in the world that take this simple question and start leading teams down the wrong path right from the start. Technical people (like the agile consultants I’m thinking of) tend to jump right to the technical things. They may say it is important to have a good understanding of the various new roles on the team. Or it is important to know how long your iterations will be. Or it is important to know various developer practices to become more productive. Or it is important to know ____ (fill in the blank with your favorite item). My feeling is that all of those things could be important, but **none of them mean anything if a team does not know its velocity per iteration!**

If a team does not know its velocity, how will that team ever be able to know how much work to put into an iteration? If a team doesn’t focus on determining its velocity, it is a certainty that the team will miss the scope of most iterations. And then what good is going Agile?

read the post at
blogs.netobjectives.com

Lean-Agile Straight Talk

NetObjectives

a podcast series

[blogs.netobjectives.com/
category/podcasts](http://blogs.netobjectives.com/category/podcasts)

COMPLETING THE AGILE DEVELOPMENT PUZZLE

by Jim Trott
May 09, 2007

Hartman is Vice President of Business Development and Marketing. He has over 20 years experience in the software industry and has seen it all. Maybe it is all those years in the trenches or maybe it is the gray in his beard or maybe it is living in Colorado, but I find his perspectives to be refreshing. He sees what organizations truly need and does a great job helping them.

Recently, I had the chance to talk with Bob just after he gave a seminar called "How to use lean principles to complete the Agile development puzzle." This seminar was motivated by Bob's keen awareness that Agile – as it is usually taught – is not nearly as effective for teams and organizations as it should be. Teams only go so far and are left to struggle with how to improve, or must hire expensive consultants. Lean helps complete the picture.

[listen to the podcast](#)

The Ascension of TDD (cont)

(Continued from page 3)

process. When errors are discovered, an inquiry as to why the error occurred should be conducted. This inquiry focuses on how the process let the error happen. Once discovered, this fault in the process should be corrected.

Many errors in software development occur due to miscommunication between developers and customers. Hence, QA must get involved where this communication takes place. When requirements are specified, this is a great opportunity to use the specification of acceptance tests to minimize the potential for errors in this communication. Our process of specifying acceptance tests at this point of failure improves the communication and therefore lowers the number of errors that will occur.

I call this attitude of using Acceptance TDD and Functional TDD to improve process, *Lean TDD*. It is an overarching attitude for how to use TDD in the process of software development.

(Continued from page 1)

play the role of the ScrumMaster, all the members need to know what Scrum is and what is expected of them. Our studies show that having the team take this together greatly reduces the time it takes before they become effective.

This course is a team-centered offering that teaches a develop-

A TDD for All Reasons

There are at least five types of error n software development:

1. The developer didn't implement the code as she understood it
 2. The developer implemented the code as she understood it, but it wasn't what the customer meant
 3. The developer implemented the code as she understood it, it was what the customer said, but now the customer realizes it wasn't what they meant.
 4. The developer implemented the code as she understood it, it was what the customer meant, but now the customer realizes it wasn't what they wanted.
 5. The developer implemented the code as she understood it, it was what the customer meant, it also was what the customer thought they wanted at the time but now want something else.
- I call all of these "defects." We should not be particularly worried about whether it is a bug, a miss-communication or whatever. The point is that value has not been provided. Lean says

not to assess blame to part of the team (saying a bug is the fault of the developer, for example). It is the team's responsibility to provide value. And the team is accountable when they don't accomplish this.

Functional TDD focuses on the first two types of defects. Acceptance TDD assists types 2, 3 and 4. All types are assisted through short iteration lengths (and which all assist in turn by allowing for automated regression testing). Lean TDD improves the entire process.

Summary

Test-Driven Development is a powerful practice to improve software development.

Functional TDD is uses unit tests to describe the functionality about to be implemented by the developer.

Acceptance TDD is uses acceptance tests to describe the stories users want developed.

Lean TDD uses the results of the test/development process to improve the process itself.

Implementing Scrum for Your Team (cont)

(Continued from page 1)

ment team how to implement Scrum. It is a combination of interactive lecture with a significant amount of time spent on hands-on exercises.

In this class, the team will learn what Scrum is, how to manage Scrum projects, how to use Planning Poker to do story estimation, the roles of the Scrum and what each does, and the

limitations of Scrum.

Additional courses in this series focus on equipping the Scrum Master and the Product Owner with additional specific skills they require to succeed.

For more information, see

[www.netobjectives.com/
services/scrum](http://www.netobjectives.com/services/scrum).

Coaches' Corner

by Alan Chedalawada

Background

The client was a large consumer products organization that does not have a formal methodology. They have had some training in use cases for analysis, but no training in Agile processes. They have selected a project that involves upgrading a user interface for an existing backend order processing system. The project must interact with other systems, which will not be following Agile processes.

The goal of this consulting engagement was to help a team that is used to informal processes and standards that vary by project and domain area to learn to incorporate the disciplines of Agile into their software development practices. Management would like to see more discipline while developers see Agile as permission to continue doing whatever they think is best without formal standards.

I was brought in a bit late, right before the start of the first iteration. The team had to get up and running quickly.

Coaching Challenges

No common standard. The client does not have a common standard for development. This is good because there are no methodology battles to fight and not need to translate Scrum into another method; on the other hand, projects are used to doing their own thing and do not like discipline..

Project prioritization. There is not a single voice for project priorities. Getting the “decision by committee” can be very awkward.

Integrate with other, non-Agile projects. The team and the project must interact with other projects that are using waterfall approaches. They do not understand and possibly resent these Agile approaches.

No time to assess. There is no time to assess the organization. The team is ready to go, so they must be trained on-the-job by the consultant.

What I Did

This was not ideal conditions for an initial engagement. I was brought in a bit after the fact. I did not know their knowledge of Agile. The project had been selected and the team had been selected. I would have pushed back, stopped the project then and there except that for this client, the rewards of future, on-going work seemed good and the risks were probably acceptable. The fact that they are fairly loose

and informal in their approaches meant that this would not be a super high visibility project. If it failed, I could use it as an example of “failing fast” and move on to the next project, using training first.

Setup iteration. I scheduled three days as a “setup iteration” where we could get things organized: infrastructure, tool set, technology, process, and standards. From a consulting perspective, I used this time to understand the team and their needs and then to perform some quick, initial on-the-job training.

First iteration. The first iteration (and a few after that) required me to teach them more. They still had not been trained. What was essential for them to know to get started? I covered the following:

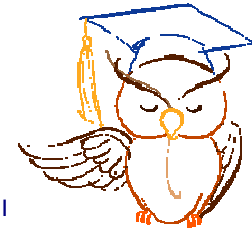
- Story sizing.
- Story-based testing up-front
- Swarming.
- Standard work.
- Product Owner responsibilities
- Retrospection.

Next iterations. In the next iterations, there is more they need to learn. Unfortunately, they did not have the training yet. So, I had to treat each iteration as another on-the-job training event for them. What is this team ready to learn next? How far can I push them?

Elevation iteration. The features developed in this project must be integrated with other products, which are using more traditional waterfall approaches and their own testing methods. We decided as a team to create an “elevation iteration” to work on the integration with the other products. The stories in iteration are all focused on integration.

Summary

Overall, did this approach take longer? Probably it required an additional iteration or two than if they had had gotten the training at the beginning. There is probably more they should learn. It forced me to scramble a bit more. But in the end, the team has gotten its work done and they are already thinking about the next engagement with me. And that seems like success to me



What would you have done? Share your ideas with me and I will post them in the next newsletter. Write me at alan.chedalawada@netobjectives.com



275 118th Avenue SE
Suites 115 and 125
Bellevue, WA 98005

Phone: 425-688-1011

Fax: 425-642-8202

E-mail: info@netobjectives.com

www.netobjectives.com

Strengthening your organization's software development processes, practices, and techniques through training, coaching, and consulting

© 2007. Net Objectives, Inc. All Rights Reserved.



Training in Sustainable Product Development

Net Objectives offers the most comprehensive Lean-Agile training in the world. Our approach is a blend of principles and practices to provide a complete team and/or enterprise wide training solution.

Assessment Services

An effective way to embark on an enterprise level transition to Lean-Agile methods is to start with an assessment of where you are, where you want to go and options on how to get there that are right for you and your budget.

Lean-Agile Coaching and Mentoring

Coaching is often the most effective way of assisting a team in transitioning to a more effective software development process. Our coaches work with your teams to provide guidance in both the direction your teams need to go and in how to get there.

Lean-Agile Mentoring

Our trainers and coaches can work on your team in a full-time capacity providing hands-on guidance on the Lean-Agile processes and practices you are adopting. This can be very competitive when compared to other companies' technical consulting/staffing services.

Coming Up

Public Courses

Bellevue, WA

Lean-Agile Testing Practices – June 26-27
Agile Estimation and Analysis for Developers and Product Owners – July 18-19
Lean Software Development – August 1-2
Design Patterns Explained – August 6-8
Advanced Software Design – August 9-10
Design Patterns Explained – October 22-24
Advanced Software Design – October 25-26

San Francisco, CA

Lean Software Development – July 26-27

SEE WWW.NETOBJECTIVES.COM/COURSES

Free Seminars

Bellevue, WA

Analysis and Architecture in Agile Projects: Seeing the Big Picture while Building Small Pieces – June 14

Denver, CO

Introduction to Sustainable Product Development: Using Lean-Agile Methods to Improve ROI – June 14

Bay Area, CA

Emergent Design: Design Patterns and Refactoring for Agile Development – June 28

Introduction to Lean Software Development – July 12
The Business Value of Pair Programming – August 15

SEE WWW.NETOBJECTIVES.COM/FREE-SEMINARS