NET OBJECTIVES EZINE
VOLUME 2, ISSUE 2 MARCH 2005

# Agile Software Development Explained

*Alan Shalloway, Net Objectives, alshall@netobjectives.com*

*To be a chapter in his upcoming book Agile Software Development Explained: A Manager's Guide, scheduled to be published in early 2006*



## Overview

### *Introduction*

We're assuming that those people reading this book have experience in software development. This is not an academic text – a book about how software development should be done if things worked right. It is about what works and what doesn't work in the real word of developing software. It mostly focuses on commercial software development – both by IT organizations and software development companies. However, our experience shows it applies to virtually all software development (engineering companies, universities, scientific research centers, etc…).

What causes software development projects to be flawed – to cost more than we think they should, to be of lower quality than we want, to be harder to maintain than we feel they ought to? Is it our people? Is it our process? Is it our lack of process?

We could get into several different discussions here. Unfortunately, most of them would be academic, or at least, not make any practical difference. The real question is – is there a way to determine how good (or bad) our software methods are and are there ways to improve them if they are not good enough?

In looking at the question of how good our software methods are, I think it is important to look at the waste in our current process. We'll wait until later to give a more formal definition of waste. At an informal level, however, I'll suggest the main areas of waste in developing software today are:

- many features are developed that are never (or rarely) used
- changes occur that result rework at a high cost
- code is hard to modify in later releases causing waste
- much of the work done in analysis is not useful
- testing software is very labor-intensive
- different development groups within the same organization re-build the same components or very similar versions of them (i.e., the actual reuse between groups is much less than the potential reuse)

*Articles and events of interest to the software development community*

***In This Issue:***



NET OBJECTIVES
*Address:*
*275 118th Avenue SE*
*Suite 115*
*Bellevue, WA 98005*
*Telephone:*
*(425) 688-1011*
*Email:*
*info@netobjectives.com*

*Volume 2, Issue 2*
*March 2005*


NET OBJECTIVES
Address:
*275 118th Avenue SE*
*Suite 115*
*Bellevue, WA 98005*
*Telephone:*
*(425) 688-1011*
*Email:*
*info@netobjectives.com*

## *Agility In A Nutshell*

When many people hear the term "agility", they think of it in terms of what *doesn't* get done when you are "agile":

- not much planning
- not much analysis
- little or no design
- no documentation
- not paying attention to risks
- not much communication to management
- no metrics

Actually, all of these things do happen. The false impression that they do not comes from two sources:

1. Some people have "adopted" agility as an excuse to "hack" and are actually not agile at all, thus presenting a poor image of agility, and
2. Much of this "agile" work happens in an integrated fashion and occurs over a longer period of the project (that is, it is spread out over time) and so the agile activities are not as apparent as non-agile activities are in standard processes

Many people also think agility has little process – this is also not true. The truth is that agile teams have a well-defined process, conduct significant planning, do lots of analysis, lots of design, produce as much documentation as is asked for, pay attention to risks, and continuously informs management as to what is happening using the best, most accurate metrics of all (I'll discuss this a little later in this chapter).

What I'd like from you, the reader, is to temporarily dispense any pre-conceived notions you may have about why agility can't work and allow me to show you what agility really is.

## *What is Agility?*

Agility is based on the realization that your team is skilled, motivated, professional, and can make intelligent decisions. While not freed of process, they are guided – not imprisoned -- by it.

## *Agility is about Focus*

The first and foremost rule about agility is to focus on the most important things in the project first. This means several things, among them:

1. Build those sections of the software that will enable customers to understand what they truly need before creating software of less value
2. Pay attention to risks that might derail or delay the project
3. Keep quality high, as not doing so will slow things down later and make maintenance and new releases much more costly

## *Agility is about seeing reality*

In addition to this focus, there is an ongoing commitment to making sure that the right work is being done and that the team knows how much work is being done. This requires:

1. Frequent contact with the customer team (this may be the actual customer or someone playing this role)
2. Frequent validation that what is being built is what was wanted
3. Frequent verification that what is being built works properly
4. An accurate assessment of the status of the project

### About the author -- Alan Shalloway

Alan Shalloway is the CEO and senior consultant of Net Objectives. Since 1981, he has been both an OO consultant and developer of software in several industries. Alan is a frequent speaker at prestigious conferences around the world, including: SD Expo, Java One, OOP, OOPSLA. He is the primary author of Design Patterns Explained: A New Perspective on Object-Oriented Design and is currently co-authoring three other books in the software development area. He is a certified Scrum Master and has a Masters in Computer Science from M.I.T.

### *Yes, there is a catch*

The ways to accomplish all of these things is relatively straightforward – build the system in small chunks. Unfortunately, this results in a different set of problems than we are used to:

1. It requires rework of code as new ideas are discovered
2. It requires retest of modules as existing code is modified

The good news is that we can limit the cost of this rework and retest. A relatively new discipline of programming called Test Driven Development (TDD) has been created to allow for incremental design in a fairly efficient manner. To be sure, there is waste in this process, but in the big scheme of things there is less waste in this process than in standard processes.

### *Waste is inherent*

*Less* waste you say? How is waste acceptable at all?

In truth waste is inherent in any process. The trick is to manage it and make it as small as possible. A simple reflection proves that our current techniques (including agility) are not 100% effective. A 100%, as-fast-as-possible technique, would be to have an analyst talk to a programmer and then, as fast as the programmer types, the code would get written (well-documented of course) and then everything would just work. The first 5 or 10 times we would run tests to make sure, but after these always worked, we'd decide we don't even need to do that anymore (remember there would be no waste so there would be no bugs).

Pipedream we know. It ain't gonna happen. There is waste in a process, what we want to do is *minimize* this waste.

## What Does Agility Look Like?

### *Incremental, Iterative, Integrated*

The main distinction of Agile development is that it builds software in incremental, integrated steps. Each step builds on the earlier one. We call this *iterative* because we are not developing the software in pre-defined chunks, but rather in steps – where each step is defined by the previous one. Finally, it is *integrated* because each chunk has some analysis, design, code and test in it. In other words, instead of what we have in Figure 1 (a classic waterfall approach) we have Figure 2 (an agile approach).

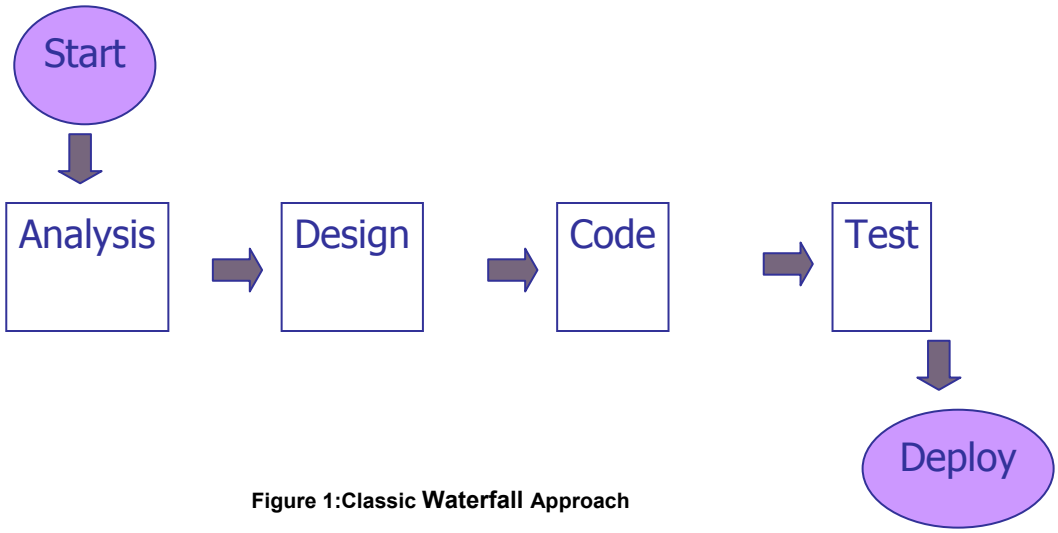



**Figure 1:Classic Waterfall Approach**


Volume 2, Issue 2
March 2005

NET OBJECTIVES
Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

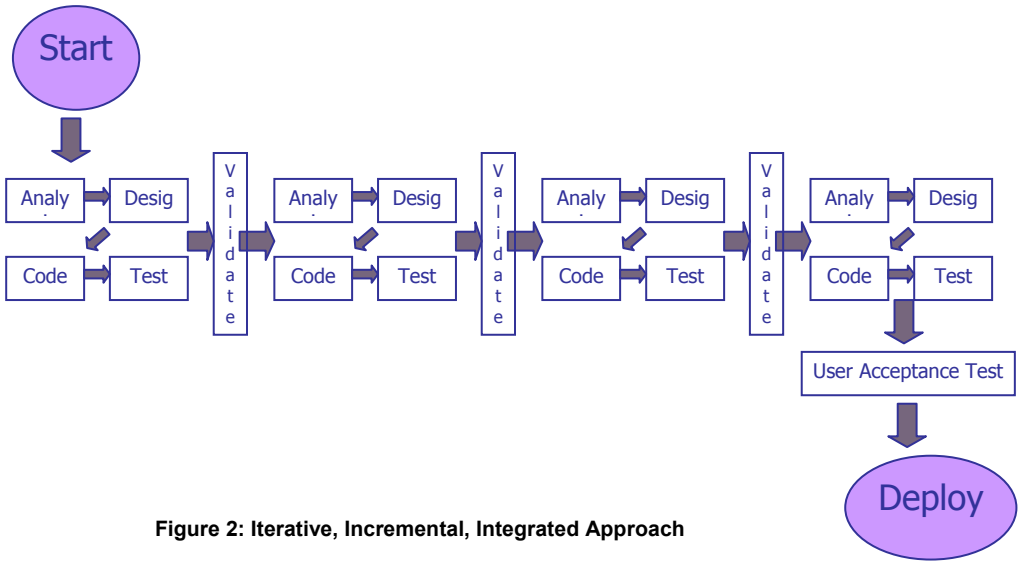




**Figure 2: Iterative, Incremental, Integrated Approach**

### *Assumptions about Agile Corrected*

Each increment is a mini-cycle including all of the work one expects to see in software development. Let's investigate agility by taking a look at each one of those items that many people *think* are left out of agility, to demonstrate that they are not:

**Some people think that in agile software development there is not much planning.** Quite the opposite. Each iteration (1-4 weeks in length) has a full day of planning to start if off. This day is called the planning game.

**Some people think that in agile software development there is not much analysis.** While there is probably less analysis done in an agile project, it is both more effective and efficient. Analysis is done in a "just in time" fashion. As much analysis is done as needed, but you don't do analysis on things you don't eventually ever work on, which happens in most waterfall projects. This is what makes it effective. It's efficient because the developers do the detailed analysis reasonably near when they actually use it.

**Some people think that in agile software development there is not much design.** This situation is similar to analysis. While minimal design is done up front, it is also done continuously. The techniques of Test-Driven Development and Refactoring actually have you designing your code most of the time. True, there is not a tremendous amount of design up front. By staging it over time, the designs that unfold are much better because they are based on what is really happening in the code, and what we learn about the domain as we go. The concerns of writing designs that are not flexible can be overcome by using design patterns and an approach called commonality – variability analysis. These two techniques, when used together, can make code very capable of being easily modified to handle new requirements.

**Some people think that in agile software development there is no documentation.** Nothing in an agile project is created unless the customer asks for it or it is a required step in order to achieve something the customer has asked for. If the customer wants documentation, all they have to do is ask for it – and documentation will be produced. What is left out of an agile project is useless documentation. Documentation that is produced "just because" but never used.

**Check www.netobjectives.com/events/pr_main.htm#FreeSeminars**
**for Public Seminars in**
**Western Washington State, Midwest, and Northern California**

## Seminars We Can Give

**Agile Planning Game** – The Planning Game was created by Kent Beck and is well described in his excellent book: Extreme Programming Explained. Unfortunately, the Planning Game as described is not complete enough - even for pure, XP teams. This seminar describes the other factors which must often typically be handled.

**Comparing RUP, XP, and Scrum: Mixing a Process Cocktail for Your Team** - This seminar discusses how combining the best of some popular processes can provide a successful software development environment for your project.

**Design Patterns and Extreme Programming** – Patterns are often thought of as an up-front design approach. That is not accurate. This seminar illustrates how the principles and strategies learned from patterns can actually facilitate agile development. This talk walks through a past project of the presenter.

**Pattern Oriented Development: Design Patterns From Analysis To Implementation** – This seminar discusses how design patterns can be used to improve the entire software development process - not just the design aspect of it.

**Transitioning to Agile** – More and more companies are beginning to see the need for Agile Development. In this seminar, we discuss what problems agility will present and how to deal with these.

**Test-First Techniques Using xUnit and Mock Objects** – This seminar explains the basics of unit testing, how to use unit tests to drive coding forward (test-first), and how to resolve some of the dependencies that make unit testing difficult.

...

**Check www.netobjectives.com/events/pr_main.htm#UpcomingPublicCourses**
**For a complete schedule of upcoming Public Courses**
**in Western Washington State, Midwest, and Northern California**
**and information on how to register**

**www.netobjectives.com • mike.shalloway@netobjectives.com • 404-593-8375**

*Volume 2, Issue 2*
*March 2005*


NET OBJECTIVES
Address:
*275 118th Avenue SE*
*Suite 115*
*Bellevue, WA 98005*
*Telephone:*
*(425) 688-1011*
*Email:*
*info@netobjectives.com*

*Volume 2, Issue 2*
*March 2005*


NET OBJECTIVES
Address:
*275 118th Avenue SE*
*Suite 115*
*Bellevue, WA 98005*
*Telephone:*
*(425) 688-1011*
*Email:*
*info@netobjectives.com*

**Some people think that in agile software development attention is not paid to risks.** Agility, when done properly, is very risk-sensitive. One of the largest risks in software today is that requirements will change or that the customer won't like what is developed. By building the software in an incremental fashion, agility mitigates this risk inherently. Other risks are identified and handled in different ways. For example, software development spikes which are used to check out a potential problem are common occurrences on an agile project.

**Some people think that in agile software development there is not much communication to management.** On the contrary, in an agile project, through the use of information radiators, management can see on a continuous basis where a project is. An information radiator is a device that shows anyone, anytime (and in some cases, anywhere) what is going on in a project and what its current status is. A common technique it to use set of stickies on a wall in the team room that shows which tasks are in the current iteration and the status of each task. A manager need only walk into the room to get a good idea of what is happening. There are software information radiators as well.

**Some people think that in agile software development there are no metrics.** Agility focuses on the two most important metrics there are:

1. How fast are you building software
2. How much more do you have to build.

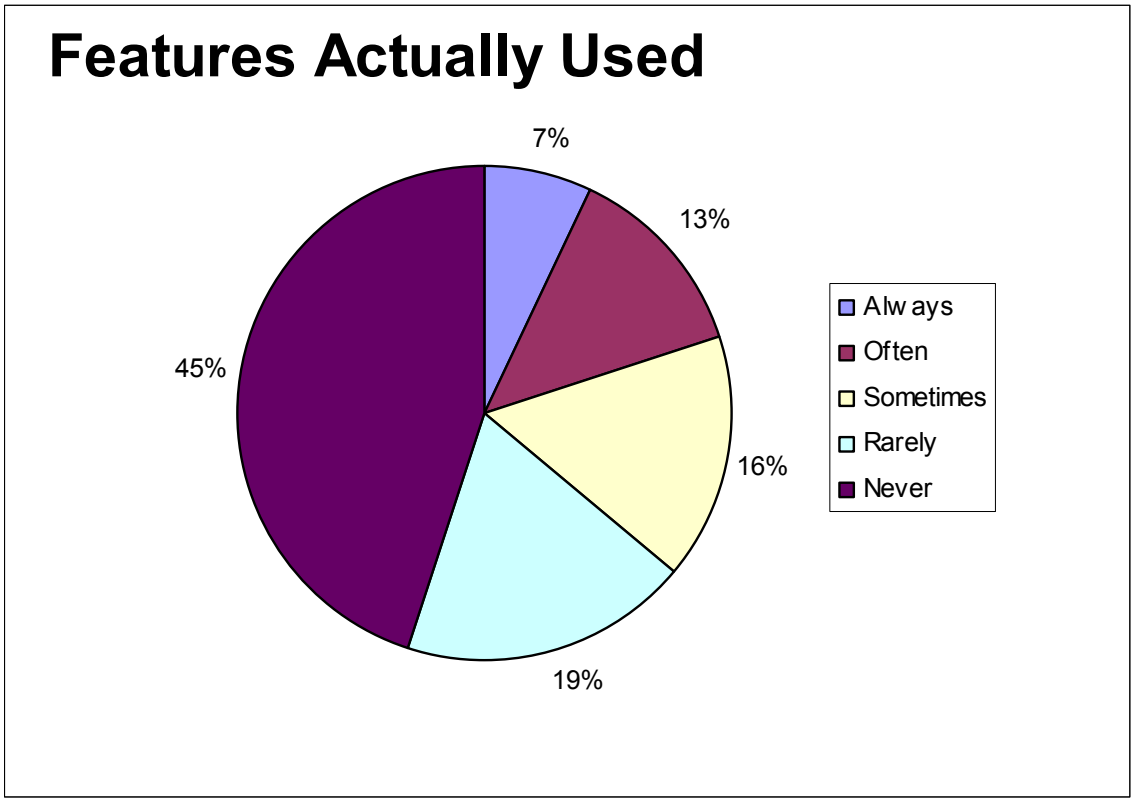
I would contend that without these, you can't tell how long a project will take. Waterfall doesn't address these metrics directly. Instead, it communicates a project's status by the use of artifacts which may or may not be related to progress (that's how you can be on time after 8 months, but be 3 months behind schedule a month later – you weren't really on time). Agilists use the amount of code developed as a true measure of where the project is. As far as I know this is the only true measure.

## The Motivations for Agility

### *Requirements will change*

I want to end this chapter with what I consider to be, at least at first, a surprising statistic – maybe even unbelievable. Upon reflection, we'll understand why it actually should be expected. Look at the diagram below:

This is from the Standish Group's study of thousands of different software projects. If you believe it, it says that 64% of all features implemented are either never or only rarely used. Although this is an astounding indictment of our industry, on reflection it is not surprising.




Let's look at why this might happen:

**Devs pick what to do.** In many software projects, the analysts discern from the customers what is to be done. After handing this information off to the devs, the devs decide what to build based on architectural issues, difficulty of development, etc. However, whether the customer will actually use a feature or not is usually not a consideration. The customer asked for it – of course they'll use it. However, let's consider what happens when a project falls behind schedule and function is paired. Chances are good that much of the function that would have been stripped from this release as not being the most valuable will have already been implemented.

**Customers change their minds.** A large software system is fairly complex. Expecting a customer to know exactly what features they will use before any of the system is built is an unrealistic expectation. That is why we have second releases (and third ones, fourth ones, etc…) and updates, etc. These releases often supplant functionality that was provided in earlier ones – thus making these earlier features unused.


Volume 2, Issue 2
March 2005

NET OBJECTIVES
Address:
275 118th Avenue SE
Suite 115
Bellevue, WA 98005
Telephone:
(425) 688-1011
Email:
info@netobjectives.com

*Volume 2, Issue 2*
*March 2005*


NET OBJECTIVES
Address:
*275 118th Avenue SE*
*Suite 115*
*Bellevue, WA 98005*
*Telephone:*
*(425) 688-1011*
*Email:*
*info@netobjectives.com*

## More Seminars We Can Give

**Agile Use Cases** - Capturing functional requirements with Use Cases is a software development best practice. Agile development processes are proving themselves to be effective. They work very well together by combining the iterative nature of agility with incremental development of use cases, as this seminar explains.

**Agility and Ceremony: How Can They Co-Exist? –** This seminar provides a short overview of Agility and the challenges one faces in a High Ceremony environment.

**C# for Java and C++ Developers –** This seminar will give a complete overview of the major language features of C#, and will also examine some best-practice issues.

**Design Patterns in an Agile (even XP) Environment – The Object Pool –** This seminar presents a project done by following the guidelines of Design Patterns, Agile Development, and Refactoring to show how these ideas can inform each other.

**Effective Coding Practices –** This seminar introduces three straightforward coding practices that support design patterns, refactoring and test-driven-development: Encapsulating Construction, Coding by Intention, and Considering Testability before Coding.

**Effective C# –** This seminar is intended for programmers who are not object-oriented experts and who want to learn how to use C# effectively.

**Emergent Design: Design Patterns and Refactoring for Agile Development** - This seminar illustrates why design patterns and refactoring, which can seem opposed to each other, are actually two sides of the same coin.

**Introduction to Use Cases –** In this seminar we present different facets of the lowly Use Case; what they are, why they're important, how to write one, and how to support agile development with them.

**The Need for Agility –** This seminar is designed to demonstrate how to balance the necessity for up-front analysis and design with the need to get feedback about how the project is going.

**Net Objectives Panel Discussion on Agile Software Development -** In this special presentation, key members of the Net Objectives training staff give their views on agile software development and take questions from the audience.

**Unit Testing For Emergent Design –** This seminar illustrates why design patterns and refactoring are actually two sides of the same coin.

**The XP Planning Game and Iteration Retrospectives –** Tools, techniques and tricks that aid iteration planning and team learning will be discussed.

and 3 on key XML topics:
**XML Data Binding With Castor**
**XSLT - Step By Step**
**The JDOM Alternative For XML Parsing**

If you are interested in any of these offerings, if your user group or company is interested in Net Objectives making a free technical presentation to them, or if you would like to be notified of upcoming Net Objectives events, please visit our website, or contact us by the email address or phone number below:

**www.netobjectives.com • mike.shalloway@netobjectives.com • 404-593-8375**

**Developers get creative.** Developers usually come up with cool ideas. Unfortunately, they often don't ask the customer if they are useful. Creating cool stuff that doesn't provide business value is called waste.

**Customers can't prioritize well.** Developers know that the things the customers ask for have different degrees of importance. A common attempt to build the most important items is to ask the customers to provide a prioritization of the requested features. "A" for essential, "B" for important, "C" for nice to have. The customer knows that the C's are very unlikely to ever get built. This may influence his decision on labeling. Unfortunately, since the things they ask for are needed for their work, it often looks like everything is an A. To validate this statement, just look at your customer requirements list. Asking customers to list things as A, B or C results in lots of A's. However, you can ask your customer – "what are the 5 most important pieces of functionality?" They may not pick the top five, but you can bet that what they ask for will typically be very important. In other words, customers can't prioritize the entire system very well, but they can prioritize the next few important features quite well. The importance of this will become clearer later in this book.

**The process encourages customers to ask for it all.** In a standard development process, the customer is asked for what they want at the start of the project. The puts pressure on the customer to get "it all" in the original requirements. Otherwise, they risk the chance of it not getting some things at all. Developers will also accuse customers of "changing their minds" of "not knowing what they want". Thus, the customer thinks about everything they might possibly want and asks for that. When pressed on the relative importance, they often tell the developers to "figure it out".

### *How much function do you want?*

Let's consider two hypothetical teams. Team "A" is fast and thorough. Team "B" is slow, but good with talking to customers. We're going to run a competition between these two teams. Each will work on the same project. Team "A", being as fast as they are builds all of the functionality requested. Team "B", being only about 1/3 the speed of team "A", builds only about 36% of the functionality requested. But, because they are very close to the customer, they build the *right* 36% (the always, often, or sometimes used functionality described above).

If you were having to support and maintain the software from either team A or team B which would you chose? Most people would pick team B's system. This not only raises some interesting questions about process, it raises quite a few questions about productivity. More on that later in the book.

### *Risks in software development have changed*

Years ago, the biggest risks in software development included:

- Would our environment support the software?
- Did we have the resources to develop the system?
- Could this be done with a computer?
- Would our performance be adequate?
- Would it cost too much to build?

*Volume 2, Issue 2*
*March 2005*


Address:
*275 118th Avenue SE*
*Suite 115*
*Bellevue, WA 98005*
*Telephone:*
*(425) 688-1011*
*Email:*
*info@netobjectives.com*

***"This approach has the enormous benefit that it enables you to do whatever is the best thing at the time. You cannot do any better for your customer than that."***
***– 10 November 1997 © 1997 Alan G. Carter and Colston Sanger, speaking of Agility, early on***

*Volume 2, Issue 2*
*March 2005*


Address:
*275 118th Avenue SE*
*Suite 115*
*Bellevue, WA 98005*
*Telephone:*
*(425) 688-1011*
*Email:*
*info@netobjectives.com*

## What's Happening at Net Objectives This Month

### Employee Spotlight: Rod Claar

***Net Objectives is pleased to announce that Rod Claar has joined the company this month as a Senior Consultant. Rod will present courses and seminars in the Western Washington State area.***

***Rod Claar has been designing, developing and implementing software since the late '80s. His early work was driven by a desire to learn and solve work related problems that required custom software. This led to a career change to the software industry. He spent over 10 years developing software, implementing solutions and managing software products used by the largest retailers in North America. This experience makes him especially valuable to retailers. His broad business and software development background give him the insight and experience to effectively help software teams of any size. Rod is a Certified Scrum Master and teaches Use Cases, Agile Development, Unit Testing and Test Driven Development.***

### Personal Book Recommendation from Alan Shalloway:

*The One Thing You Need to Know : ... About Great Managing, Great Leading, and Sustained Individual Success by Marcus Buckingham. (ISBN: 0743261658) Great book on leadership, management and personal success.*

Nowadays, with the pace and competition of software development having increased, the biggest risk seems to be –

> Will this satisfy the needs of the customer better than my competition?

Interestingly enough, this risk has already caused most projects to be much more agile than they ever have been before. Although my evidence may be anecdotal, it is clear that the length of software development projects is shrinking. Projects that took 2 years to complete were not uncommon 10 years ago. Now, it's difficult to get projects to take more than 9 months. The iteration length of software has dropped dramatically. We build software in stages (i.e., more agilely than before).

### *The essence of Agility*

Agile methods are centered on determining the most important items to build and then building and maintaining them in the most efficient manner. Dealing with the realities of changing requirements and customer abilities is intrinsic in the process. The details of agility will be described later in this book, but the motivations are hopefully clear now.

**To join a discussion about this article, please go to**
**http://www.netobjectivesgroups.com/6/ubb.x**
**and look under the E-zines category.**

## Summary

### *Different philosophies*

Developing software with a standard process of doing analysis, design, code, then test assumes that requirements can be discerned early and will remain relatively unchanged for the length of the software project. Unfortunately, our experience has shown us that this is not usually the case.

Agility purports to solve the problem of changing requirements. It requires the ability to do the following:

- Be in close contact with the customer
- Have a method of doing analysis that is more efficient and effective than doing it all up front would be
- Be able to build code that is readily changeable
- Be able to test code quickly

Fortunately, the techniques of the ever-unfolding story, design patterns, refactoring and test-driven development (all to be described later in this book) allow for this.

I am reminded of a common saying – "Insanity – doing the same thing over and over again and expecting different results." My company, Net Objectives, has created a t-shirt with this saying on the front and "I used to be insane – now I'm different" on the back. Let's learn how to be different. ☺

**Technical Book Recommendation:**
***Agile Project Management with Scrum (Microsoft Professional) by Ken Schwaber. (ISBN: 073561993X) Great book by one of the masters.***

*Volume 2, Issue 2*
*March 2005*


Address:
*275 118th Avenue SE*
*Suite 115*
*Bellevue, WA 98005*
*Telephone:*
*(425) 688-1011*
*Email:*
*info@netobjectives.com*


*Volume 2, Issue 2*
*March 2005*


NET OBJECTIVES
Address:
*275 118th Avenue SE*
*Suite 115*
*Bellevue, WA 98005*
*Telephone:*
*(425) 688-1011*
*Email:*
*info@netobjectives.com*

## Net Objectives - What We Do

**Net Objectives provides enterprises with a full selection of training, coaching and consulting services. Our Vision is "Effective software development without suffering". We facilitate software development organizations migration to more effective and efficient processes. In particular, we are specialists in agility, effective analysis, design patterns, refactoring and test-driven development.**

**We provide a blend of training, follow up coaching and staff supplementation that enables your team to become more effective in all areas of software development. Our engagements often begin with an assessment of where you are and detail a plan of how to become much more effective. Our trainers and consultants are experts in their fields (many of them published authors).**

**When you've taken a course from Net Objectives, you will see the world of software development with new clarity and new insight. Our graduates often tell us they are amazed at how we can simplify confusing and complicated subjects, and make them seem so understandable, and applicable for everyday use. Many of our students remark that it is the best training they have ever received.**

The following courses are among our most often requested. This is not a complete list, though it is representative of the types of courses we offer

**Agile Project Management** - This 2-day course analyzes what it means to be an agile project, and provides a number of best practices that provide and/or enhance agility. Different agile practices from different processes (including RUP, XP and Scrum) are discussed.

**Agile Use Case Analysis** - This 3-day course provides theory and practice in deriving software requirements from use cases. This course is our recommendation for almost all organizations, and is intended for audiences that mix both business and software analysts.

**Design Patterns Explained: A New Perspective on Object-Oriented Design** - This 3-day course teaches several useful design patterns as a way to explain the principles and strategies that make design patterns effective. It also takes the lessons from design patterns and expands them into both a new way of doing analysis and a general way of building application architectures.

**Test-Driven Development: Iterative Development, Refactoring, and Unit Testing** - This course teaches how to use either Junit, NUnit or CppUnit to create unit tests. Lab work is done in both Refactoring and Unit Testing together to develop very solid code by writing tests first. The course explains how the combination of Unit Testing Refactoring can result in emerging designs of high quality.

**Effective Object-Oriented Analysis, Design and Programming In C++, C#, Java, or VB.net** - This 5-day course goes beyond the basics of object-oriented design and presents 14 practices which will enable your developers to do object-oriented programming effectively. These practices are the culmination of the best coding practices of eXtreme Programming and of Design Patterns.

**Software Dev Using an Agile (RUP, XP, SCRUM) Approach and Design Patterns -** This 5-day course teaches several design patterns and the principles underneath them, the course goes further by showing how patterns can work together with agile development strategies to create robust, flexible, maintainable designs

If you are interested in any of these offerings, if your user group or company is interested in Net Objectives making a free technical presentation to them, or if you would like to be notified of upcoming Net Objectives events, please visit our website, or contact us by the email address or phone number below:

**www.netobjectives.com • mike.shalloway@netobjectives.com • 404-593-8375**