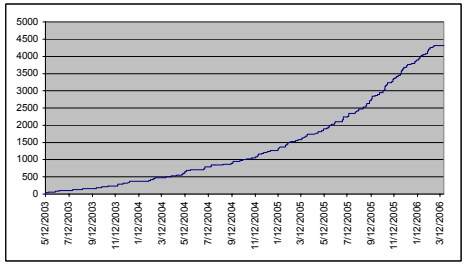
The Basic Scrum Engine
"what make agility work"

Douglas Shimp
Doug_shimp@netobjectives.com
www.netobjectives.com
info@netobjectives.com

Scrum is Growing Fast



- The need for ScrumMasters is growing fast
- Over 30 ScrumMaster Trainers (CSTs)
- Over 4000 certified ScrumMasters (CSMs)

Agenda

- Setting the Stage
  - The Goals of Software Development
  - Two Powerful Attractors
- Discussion of Scrum
  - Players/Roles
  - The Promise *of* Scrum
  - Scrum Overview
  - Notes on Scaling
  - The Promise *to* Scrum

THE GOALS OF SOFTWARE DEVELOPMENT

The Business of Software

- Software Businesses typically want two things:
  - The right software
  - Delivered fast
- Strategies
  - Make it right, hope you're fast
  - Make it fast, hope it's right
  - Some combo…
- What do you do?
- How's it working out for you?

Making it right: What You Focus on…

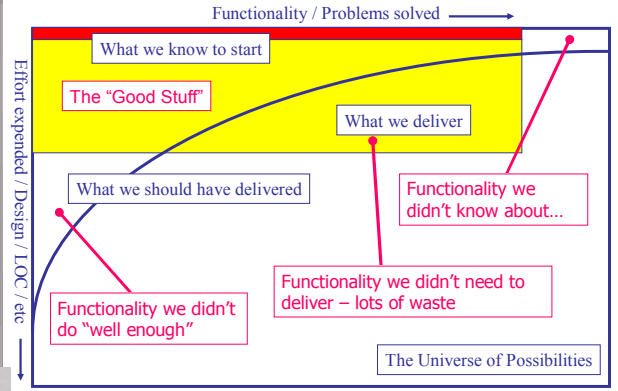
- Depends on what Stakeholders you pay attention to
- We Believe that most important Stakeholders are:
  - Ultimate users, as they determine whether the product is actually useful or not
  - Maintenance guys, as they will have to keep the product running after it's developed
  - Business owner, as it's his/her money…

So, the *Goal* of Development is…

To provide a suitable solution for users…
That consists of quality code…
And doesn't cost too much

And, do it fast

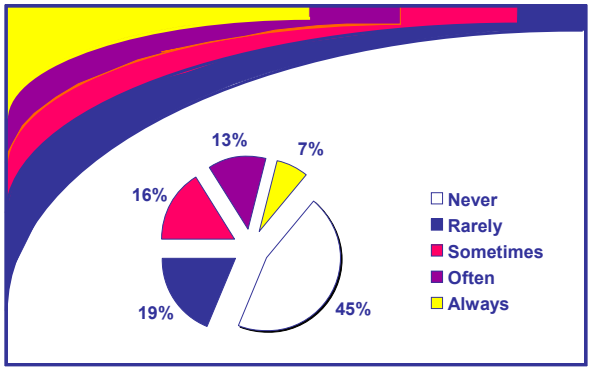
How we Normally Deliver Functionality



So, the "Ideal" Software Process would…

- convert effort you would spend unwisely
  - Non-essential functionality
  - Doing the "wrong thing"
- into developing things you don't know about yet
  - New functionality
  - Additional "robustness"
  - As-yet undiscovered bugs
- And Eliminate Waste in the Process…
- I'm not so good at looking into the future, so this requires some sort of iterative process…

Standish Study and "Ideal" Iterative Development



Summary of Software Development

- **Goal**: Provide a suitable solution for users…That consists of quality code…And doesn't cost too much
- **Ideal**: Convert effort you would spend unwisely into developing things you don't know about yet
- **Common Strategy**: Get your money, then spend it wisely
- **Implementation**: We claim this *requires* an agile process – just hold on a while longer

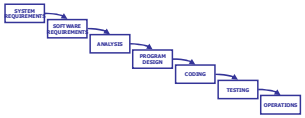
Major Risks

- Our goals are to be
  - Correct
  - Fast
- So our major risks are:
  - Delivering a product our users don't want
  - Creating waste along the way
- Scrum manages both…

TWO POWERFUL ATTRACTORS

Two Common (evil) Attractors

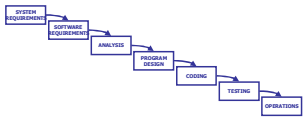
- Waterfall Process



- Hero / Lone Wolf / Cowboy Coder

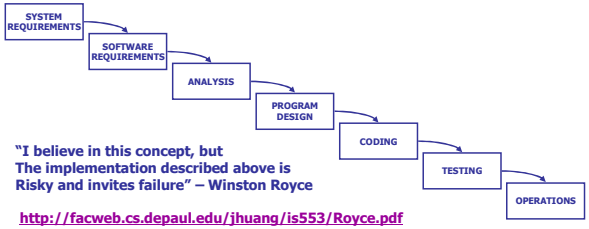
- Let's discuss each of them…

The Waterfall



Royce: the Waterfall Model

- In 1970, Dr Winston Royce published "Managing the Development of Large Software Systems"
- Ever since then he has been blamed for the "waterfall model" of software development

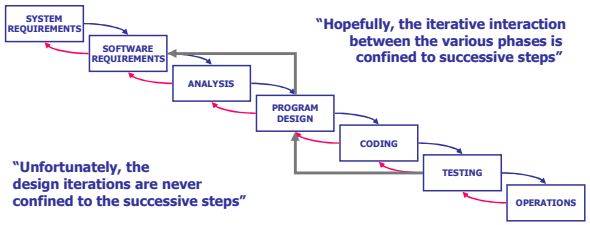


"I believe in this concept, but The implementation described above is Risky and invites failure" – Winston Royce

http://facweb.cs.depaul.edu/jhuang/is553/Royce.pdf

This Isn't Fair

- On the very next page he showed what one actually needs to do to succeed – and it was agile…



Waterfall is a Powerful Attractor

- For Management
  - It is a "defined process" – thus "feels right"
  - It gives the illusion of control
    - Of people
    - Of money
  - Allows for "hands off" management
- For Developers
  - They're just "following a recipe"
  - They're just left alone for long periods
- And it can be streamlined to be fast…

Waterfall is a "Strawman"

- Almost no projects actually do a pure "Waterfall"
- But they do base their process on *prediction* and *modification*
  - Predict what will happen
  - Modify as we go along, if necessary
- There is usually a resistance to Modification
  - Thought of as "overhead"
  - Change Control Boards
- So, while waterfall itself is not completely accurate, the problems it illustrates are

Heroes / Lone Wolves / Cowboy Coders

Heroes / Lone Wolves / Cowboys

- Cowboy Coders are programmers who write code according to their own rules. They write the “right code, the right way, but they decide what is right”
- They may be very good at writing code, but it doesn't generally follow the standards, processes, policies, or anything else derived from the group. Cowboy Coders work well alone, or in the old-style CaveProgrammer environment, but they rarely, if ever, work well in a team. Often times, they are a burr in the saddle that keeps the team from getting positive work done.
- Also called Lone Wolf or Heroes, an organization that relies on Cowboy Coders is at the lowest level of maturity, according to the CMM. Paradoxically, Cowboy Coders are often found in high-ceremony, waterfall shops… and are the reason they succeed when they do

Lone Wolves are a Powerful Attractor

- For Management
  - It’s handled: “Sue will just get it done…”
  - Absolves Management of actually managing
- For Developers
  - Everyone wants to be a hero, right?
  - They get to do things “their own way”
  - They’re just left alone for long periods
- And they can be fast, too…

Heroes / Lone Wolves / Cowboys (continued)

- Cowboy Coders often have:
  - An emphasis on ingenious artistry as opposed to plodding team-play. The cowboy approach often reaches new heights through the artistry of the practitioner. The results, however, are often spotty and rarely easy to duplicate.
  - A nearly complete absence of advance planning. However, there is a branch of the cowboy discipline that thrives on the blue-sky possibilities of software, discussing what is going to be done until the sun rises.
  - A disregard for the social aspects of cooperative programming, however, some cowboys hang out together with outstanding results.
  - A focus on "active change". When a Cowboy Coder doesn't like something, he fixes it.. RAD, Incremental Development, the Spiral Model, agility (including XP and Scrum) are all attempts to retain the aggressiveness and passion of the Cowboy while gaining the benefits of risk-aversion.

Agenda

- Setting the Stage
  - The Goals of Software Development
  - Two Powerful Attractors
- ➢ Discussion of Scrum
  - Players/Roles
  - The Promise *of* Scrum
  - Scrum Overview
  - Notes on Scaling
  - The Promise *to* Scrum

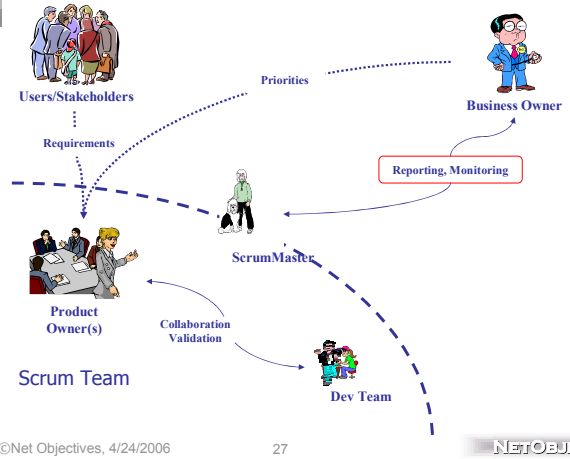
PLAYERS AND ROLES

Fundamental Players/Roles

Scrum Process

- User/Stakeholder Community
  - Has problems to be solved
  - Usually disorganized, chaotic, group
- Product Owner/Customer Team
  - Provides requirements and validation
  - Should speak with "one voice"
  - Represents all outside Stakeholders to the Team
- Development Team
  - Actually builds the stuff
  - Lots of different roles here
- ScrumMaster/Project Manager
  - Manages the process, resolves impediments
  - Responsible for the "health" of the Team
  - Interfaces with Business Owner (along with Product Owner)
- Business Owner
  - Manages resources and money
  - Often ignored in Development Process…

Relationships Between the Players



Self Organizing Team

- Scrum Teams are filled with people who have skills, not people playing roles
- The individuals on a team self-organize for the task at hand
- The basic unit is the "Teamlet", "Work Cell", or "Ideal Team"
  - The Teamlet has all the skills it needs (analysis, development, design, test, documentation, …)
  - It typically consists of 2-4 people to get all the skills "covered"
  - It swarms on one thing at a time

"good" Team Philosophies, found in Scrum

- **One Bite at a Time**: reminds us to do things a little at a time, with planning, validation, and management of the pieces.
- **Validation Centricity**: reminds us that the activities of validation, verification and test are "more important" than those of analysis, design, and construction; and that we must actively look for things that cause us to change.
- **Avoid and Eliminate Waste**: work on those things with the most value; have retrospectives to evaluate process, etc
- **Risk Sensitive**: Base decisions on risk analysis and mitigation – requirements risk, architectural risk, technical risk, quality risk, people risk, etc
- **Let the Product Lead**: decisions must be based on the product, not documented plans, analyses, requirements, or designs.

Personal Qualities we Want

- There are also personal qualities we like to see in the members of our team. We call them the "team values"
  - Play To Win
  - Communication
  - Cooperation
  - Trust

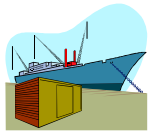
THE PROMISE *of* SCRUM

A List of Agile Processes

- There are a number of Software Development Processes that support various elements of agility
- The most common ones are:
  - Scrum – Ken Schwaber, Mike Beedle
  - eXtreme Programming (XP) – Kent Beck
  - Rational Unified Process (RUP) – Rational
- Others include:
  - Dynamic Systems Development Method (DSDM)
  - Crystal Methodologies – Alistair Cockburn
  - Feature Driven Development (FDD) – Peter Coad
  - Adaptive Software Development – Jim Highsmith
  - ...

Agility is…

- Observing and Adapting to your environment – moving faster than things that will harm your project…
- Keeping up with relevant changes…
  - In requirements
  - In priorities
  - In knowledge of our system
  - In environment
- Agility is only relevant in a context…



Things to Worry About…

- Changing Requirements
  - The Business Changes
  - New Functionality
  - Learn more about system
- Changing Priorities
  - Different Stakeholders
  - New Situations
- Changing Environment
  - New OS, Languages, etc
- Changing Budgets
  - Fewer Developers
  - Do More with Less…
- More…

- How Can We Make This Work?

Promise of Agility

- The Basic Promise is that Agility allows you to develop software successfully in the face of:
  - Unknown/Moving Requirements
    - Goal: to produce a product that is useful at the time of delivery
    - Delays analysis and design until "needed'
    - Reduces waste
  - Constrained Resources
    - Goal: to produce the best product you can with what you've got
    - Constant reprioritization
- Scrum adds the Promise that:
  - The team will adapt its processes to improve their abilities to deliver quality software

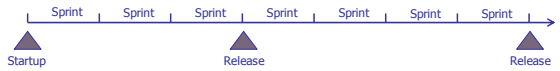
The Essence of the *Practice* of Agility is Validation and Feedback

- User / Customer Feedback Loop allows us to Satisfy our Users

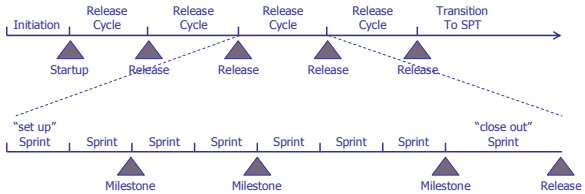
And there's another loop in here that allows us to build quality code …

Different Styles

Lightweight



Heavyweight

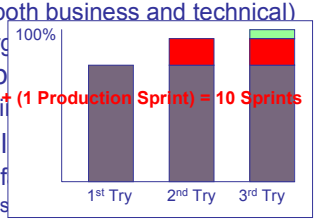


And They're Both Scrum!

Speed Versus Risk

- Our environment forces on us a certain minimum number of sprints necessary to deliver the "right" software
  - Uncertainty of requirements
  - Complexity of domain (both business and technical)
  - "Magic number is 10" arg
- Reducing number of sp
  - But increases risk of bei
- So we must be very val
  - Check to see if right so f
    - Reduce number of sprints
    - Increase them as needed

(3 Releases) x (3 Sprints/Release) + (1 Production Sprint) = 10 Sprints

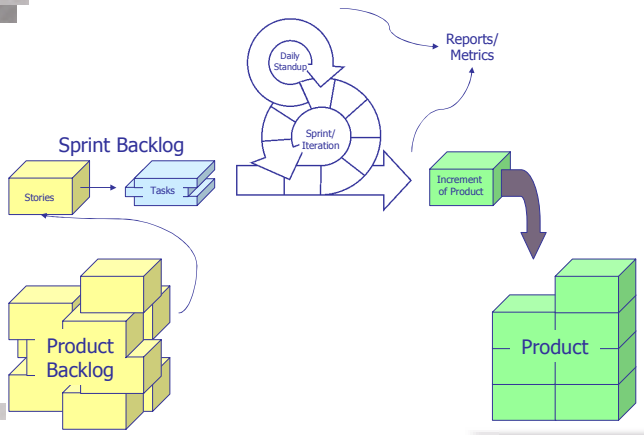


Speed Versus Risk

- Our environment forces on us a certain minimum number of sprints necessary to deliver the "right" software
  - Uncertainty of requirements
  - Complexity of domain (both business and technical)
  - "Magic number is 10" argument…
- Reducing number of sprints improves speed
  - But increases risk of being wrong
- This is where validation centricity comes in
  - Check to see if right so far
    - Reduce number of sprints on the fly, or
    - Increase them as needed

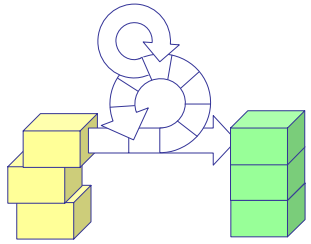
SCRUM OVERVIEW

Basic Scrum Process

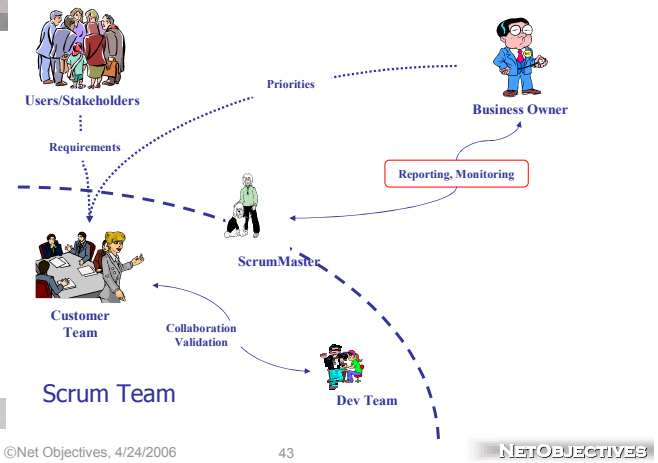


The Basic Unit is the Sprint

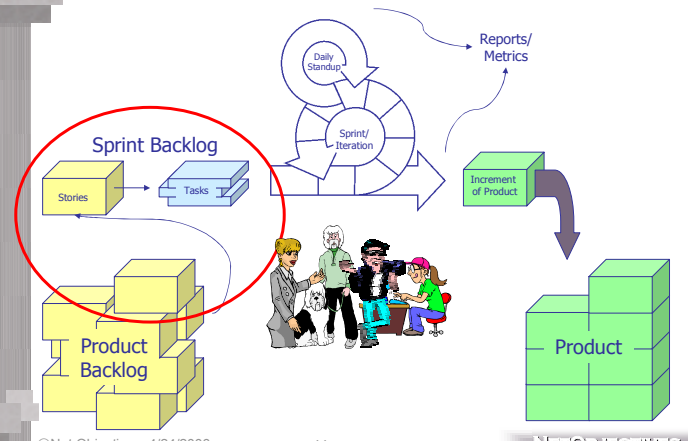
- For Each Sprint you:
  - Plan
    - What's the next stuff to do?
    - The team decides how much it can do
  - Perform
    - Demonstrable Business Value
    - Potentially shippable Product
    - Quality Process
    - Daily Monitoring
  - Evaluate
    - The Product
    - The Process
- Repeat…



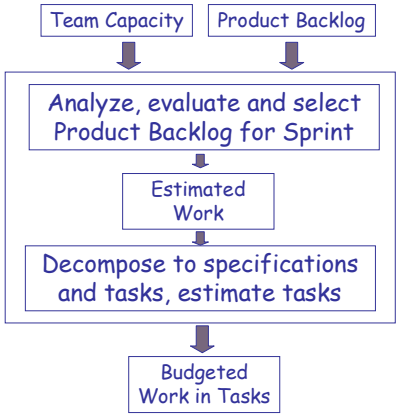
So, How Do We Make it Happen?



Sprint Planning Meeting



Sprint Planning Meeting

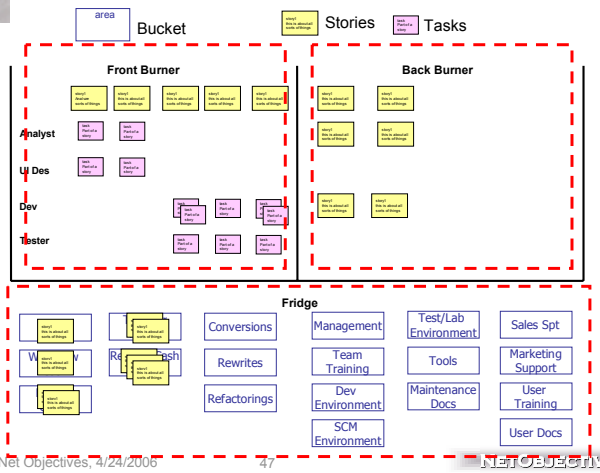


Sprint Planning Meeting Rules

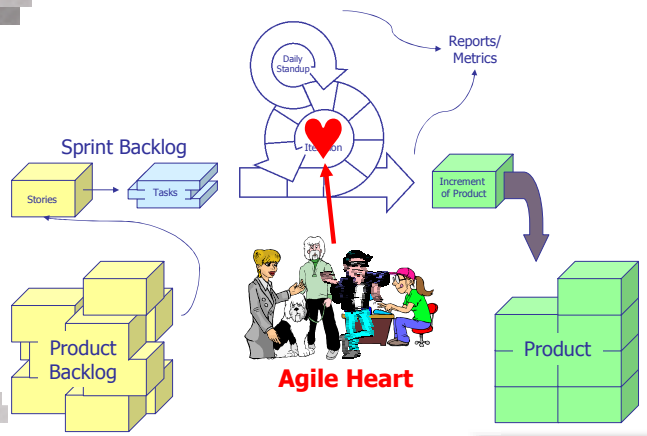
1 day

- 1st - 3 hours max. for DevTeam and Product Owner to set sprint goals and select subset of Product Backlog to work on (PO job – prioritize work todo)
- 2nd - 3 hours max. for DevTeam and OnsiteCustomers to define Sprint Backlog to build functionality (decompose to tasks and estimate)
- 3rd – 1 hour max. Get back together and decide what we will actually do

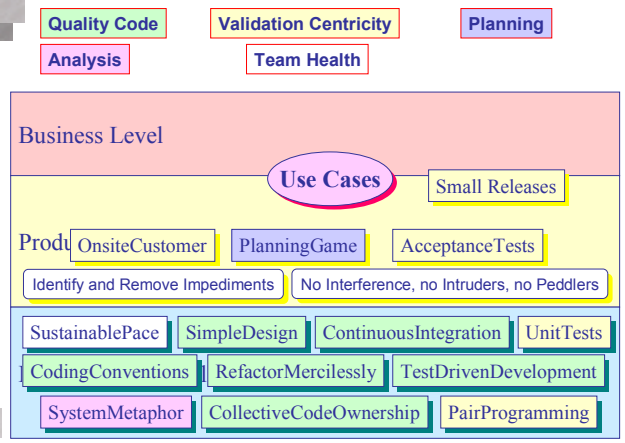
Backlog on the Wall



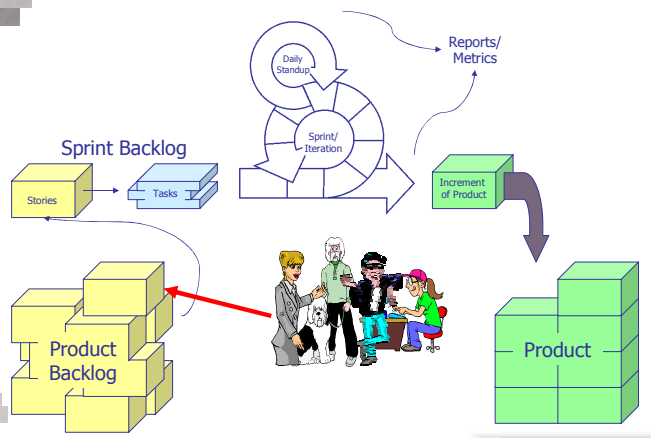
Doing the Work



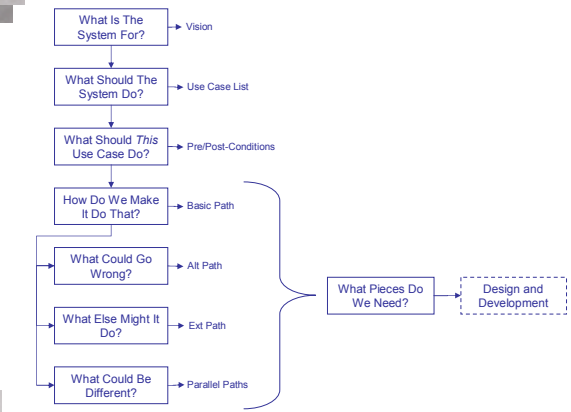
"good" Practices (XP, Use Cases, etc)



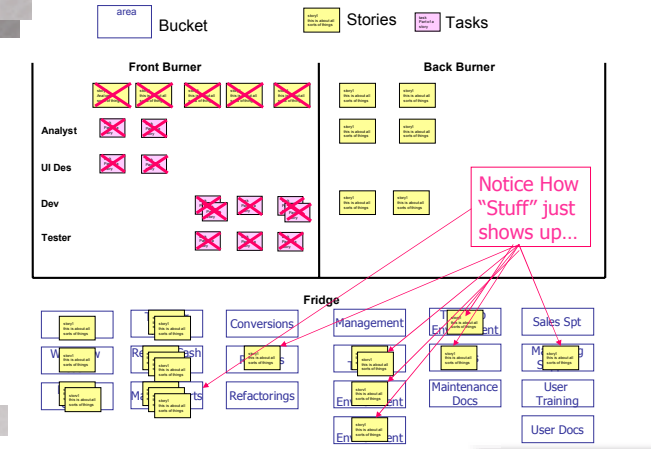
Constantly Updating the Backlog



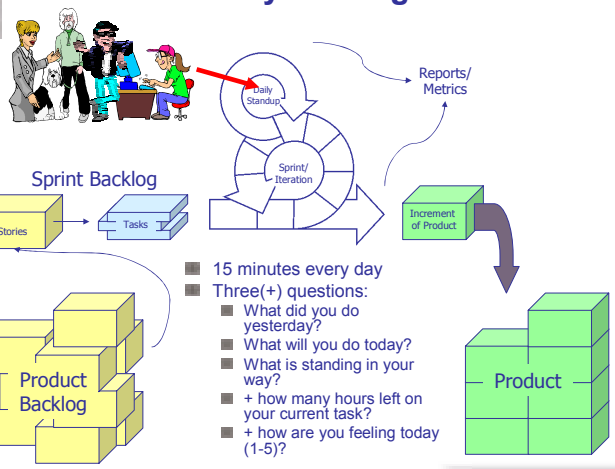
Analyzing Functionality of the System



Adding Stories to the Fridge as We Go



Daily Meeting



- 15 minutes every day
- Three(+) questions:
  - What did you do yesterday?
  - What will you do today?
  - What is standing in your way?
  - + how many hours left on your current task?
  - + how are you feeling today (1-5)?

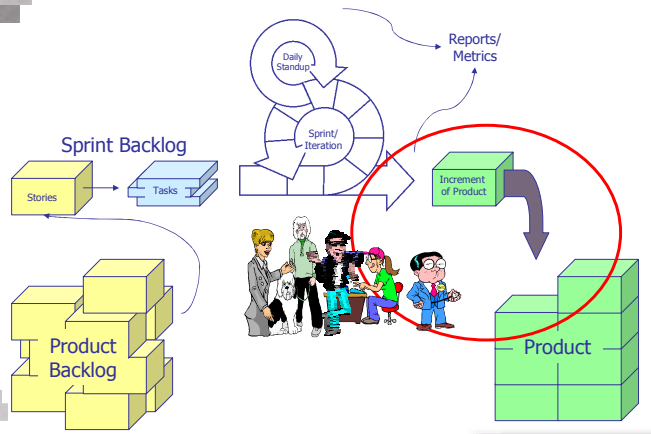
Document our Impediments

- Not enough disk space on disk subsystem server
- Build process requires too much “hand tooling”
- Joe (a stakeholder) not available for validation of xxx Story
- Sally getting yanked off for bug fixing on current release

Capture our Metrics

- Visual methods available, as well

Review and Retrospective



Sprint Review

- Finally, the Team Presents sprint results to Management and other Stakeholders
  - Demonstrate the Product
  - Present results of Functional Tests
  - Describe progress towards release and/or milestones
  - Show anything else they want to "show off"
- Note – This is an informal presentation
  - No powerpoints, just show the thing
  - Don't spend more than 2-4 hours on it…

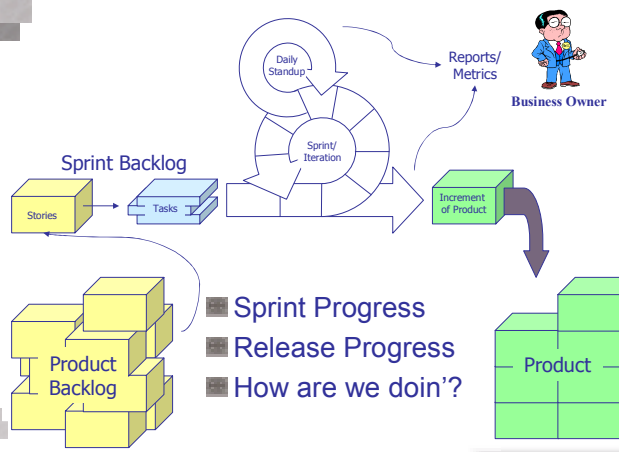
Typical Sprint Review

- The ProductOwner gives an overview
  - The Goals of the release and iteration
    - What milestones we're headed towards
    - What functionality was planned for the iteration
    - What Functional tests were used to validate it
  - Summary of Results of the Iteration
- The Developer Team gives a demonstration
  - Demonstrates the functionality
  - Emphasizes what was added this iteration
- Purpose is to provide visibility
  - Look what we did!
  - What do you want us to do next?

Retrospection

- As one last thing, the team evaluates and modifies the Process as needed
- Agility is about Feedback
  - The Retrospection is where the Team gives/gets feedback about the Process itself
- The Team can change the process
  - "If it isn't fun you're doing something wrong"
  - This allows a cohesive team to emerge
  - Must use values of communication, cooperation, and trust
  - They do this alone
  - Everyone gets a say
  - Discussion is cordial
  - Vote on a moderator/judge

Reporting and Metrics

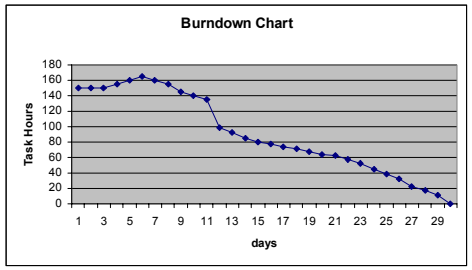


- Sprint Progress
- Release Progress
- How are we doin'?

Plan: Sprint Backlog

- Visual methods available, as well

ProjectXX Sprint Burn Down Chart



- Collect metrics daily

Status: Impediments

- Not enough disk space on disk subsystem server
- Build process requires too much “hand tooling”
- Joe (a stakeholder) not available for validation of xxx Story
- Sally getting yanked off for bug fixing on current release

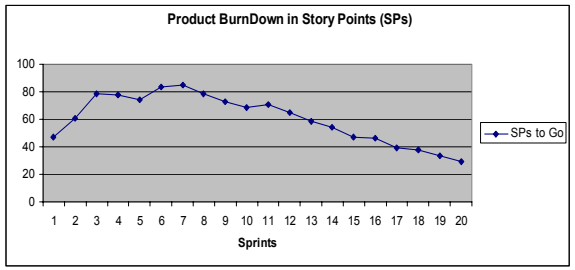
- What impediments have you seen?

Product Backlog in Spreadsheet

Feature	Story	Priority	Size
Schedule an Onsite Course			
	Schedule Onsite Course by entering all data (including dates) in text fields on a simple dialog	1	Medium
	Use “click-drag” feature of CompanyCalendar to select dates, then make them editable in dialog	2	Small
	Select ClientName from dropdown, with option to add new ClientName to database	4	Small
	Select InstructorName from dropdown, with option to add new InstructorName to database	3	Small
	Automatically add appropriate Travel Days	4	Big
	Identify Conflicts in Instructor Schedule	4	Unknown
	Select ClassName from dropdown, with option to add new ClassName to database	3	Small
	Validate that Instructor is qualified to teach the selected Class	4	Small
	Add Course to Client’s EngagementList	4	Medium
	Automatically create and Send NotificationEmails (note: many versions of this one, as more and more data is in database)	3	Medium
Implement Company Calendar			
	Be able to view activities in Calendar	1	Large
	Select OnsiteCourse and edit it	2	Small
	Implement “click-drag” to select a date range	1	Medium

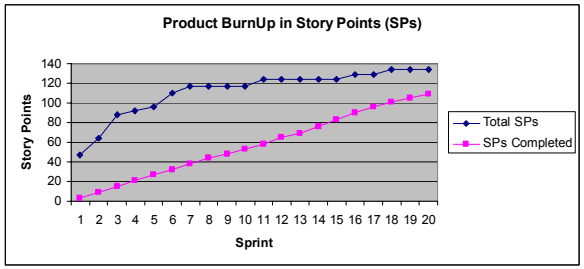
- This spreadsheet represents the WBS
  - the priorities say when they’ll get done
  - Size is a rough measure of level of effort for development team

ProjXX Product BurnDown Graph



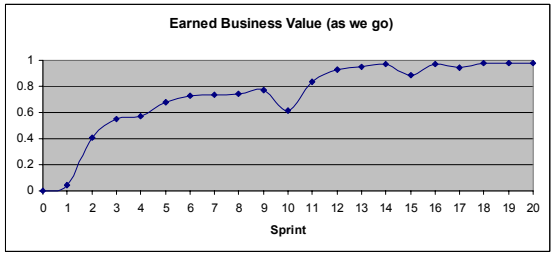
- What was going on at the beginning? Bad estimates, scope creep, what?

ProjXX Product BurnUp Graph



- The scope creep is evident in the top line
- But the velocity is pretty constant throughout – so our estimates were good
- But why did we stop with about 20 SPs to go?

ProjXX Earned Business Value (as we go)



- This shows the percentage of known business value "in the hole" at each iteration

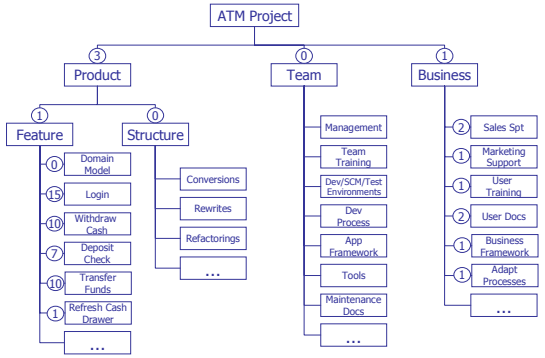
Business Value Basics

- Agility is about providing business value, and we would like a way to measure it, called BV
- Stories are units of work that provide value, so it would be nice to have a function, BV(story), that quantified the value of the story
- When a story got finished, we would "earn" its value, and this would have an Earned Business Value (EBV) for the project,

$$EBV(Project) = \sum_{done/done/done} BV(Story)$$

- The next couple slides show we can use our Feature-Based WBS to calculate such a thing…

Calculating Business Value (BV) Using a Functional WBS



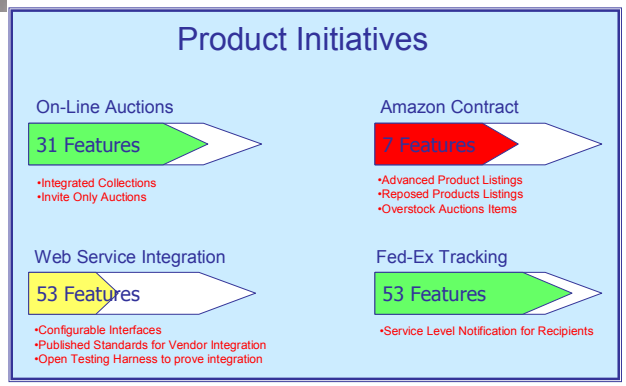
Second, We Assign Weights to the Stories Within the Buckets

- Login
  - Determine Post-Conditions for Login (wt: 0)
  - Analyze to PL2 (wt: 0)
  - Code up Basic/Happy Path (wt: 10)
    - UI mockup validated with Sam
    - Middle-tier code review
    - Functional tests validated with Sue
    - Integrate Login Module pieces
    - Unit Tests validated with Tom
    - Functional tests passed

    All checklist items must be complete before the story can earn its value
  - Analyze primary business impediments (wt: 0)
  - Code up "3 strikes and You're Out" (wt: 3)
- So, basic path of Login has BV

$$(1)\times(3/4)\times(1/1)\times(15/43)\times(10/13) \approx 20\%$$

Status: Dashboard for An Executive

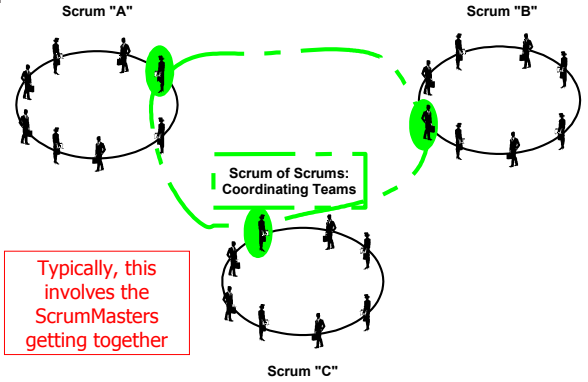


NOTES ON SCALING

Scrum of Scrums – Workgroups

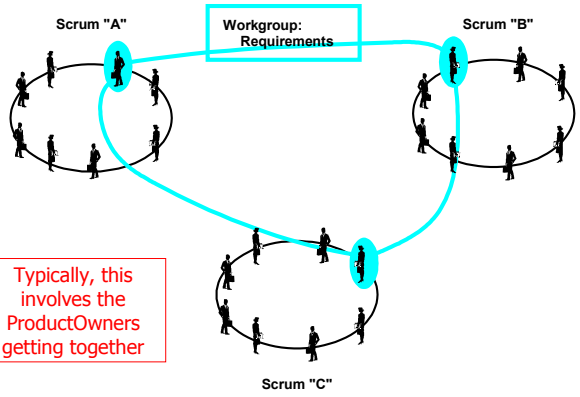
- Keep management apprised of progress across all projects
- Keep cross-cutting requirements in synch
  - Integration stories and tasks
- Keep different projects technically coordinated
  - Assist in the re-use of code
  - Common Infrastructure tasks
- Other Reasons as Needed

Scrum of Scrums – Coordinating Teams and Reporting to Management



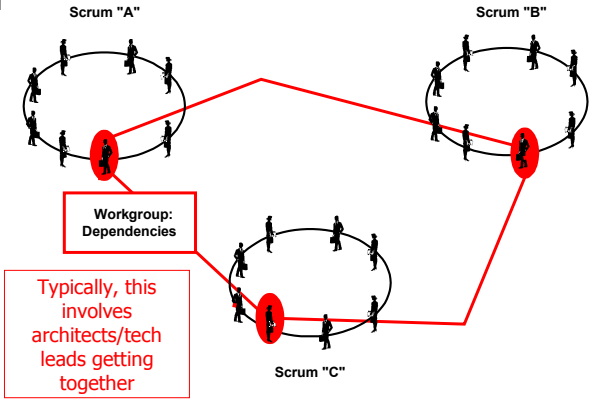
Typically, this involves the ScrumMasters getting together

Workgroup: Cross-Cutting Requirements



Typically, this involves the ProductOwners getting together

Workgroup: Technical Dependencies



Typically, this involves architects/tech leads getting together

THE PROMISE *to* SCRUM

Promises to Scrum

- In order to make Scrum work, the organization and project makes (and must keep) some promises
  - To its Stakeholders
  - To its Development Team
  - To its Customer Team

Basic Promises Made

- We promise the Stakeholders that there is a Product Owner driving the team based on their interests
- We promise the Product Owner that there are Stakeholders who will help when needed
- We promise the team that there is a Product Owner who provides an initial Product Backlog and ongoingly prioritizes the work in order to satisfy business needs
- We promise the Stakeholders that the team does quality work and delivers demonstrable product every sprint based on the Product Owner's priorities

More Promises

- We promise the team that they will be allowed to do quality work in a (relatively) uninterrupted way
- We promise that team that the priorities will not change in the middle of a sprint without their consent
- We promise the team that there are empowered Available Customers who will answer business domain questions promptly (minutes / hours, not days)
- We promise the team that there is a Scrum Master who will remove impediments and look out for their health
- We promise the team that being on a Scrum project will not hurt their careers

Note That…

- Many of these promises concern the Product Owner
- It has been my experience that the lack of an empowered, rapidly-responding Product Owner is the most common cause of failure
- The second most common cause of failure is changing priorities within a sprint (moving things in and out)
- The third one is not empowering the team to develop its own process to meet their realities…

Impact of Broken Promises

- A promise broken is a organizational/process issue, not a team issue
- For example, we don't blame a Product Owner for not being able to make decisions
  - We blame the business for not empowering the Product Owner to make decisions
  - We blame the business for not allowing the Product Owner to learn enough about the business
  - Possible Solution: The team calls "time out" and does other stuff while the Product Owner goes off and gets up to speed

Any Questions?

Please Fill Out Evaluations

Net Objectives – Who We Are

- **Vision**: Effective software development without suffering.
- **Mission**: To assist companies in maximizing the business value returned from their efforts in software development and maintenance. We do this by providing training, coaching, and consulting that both directly assits and empowers our customers to create and sustain this ability.
- **Our staff:** Includes several recognized experts in the area of Scrum, Use Cases and Design Patterns.
- Our expertise includes:
  - Lean Software Development
  - Agile methods (Scrum, XP, RUP)
  - Agile Analysis
  - Design Patterns
  - Test-Driven Development

Net Objectives' Community of Practice

- We provide a public forum for discussion of topics including:
  - Design patterns
  - Agile methods (including XP, Scrum, RUP)
  - Use cases
  - Emergent design

- We also have several books under development on-line for review by the community.
- Go to http://www.netobjectivesgroups.com for more information.
- To subscribe to our e-zine, go to http://www.netobjectives.com/subscribe.htm and fill out our online subscription form.

Upcoming Public Courses
In the Midwest

- ScrumMaster Certification – May 23-24, Chicago, IL
- ScrumMaster Certification – June 5-6, Bloomington, MN
- Managing Agile Requirements: The Product Owner's Role – June 7-8, Bloomington, MN
- ScrumMaster Certification – August 16-17, Bloomington, MN

See http://www.netobjectives.com/events/pr_main.htm for more information

Overview of Training

- **Lean Software Development**
  - **Lean Software Development: A Practitioners Course**
  - **Lean Software For Managers**
- **Agile/Scrum**
  - **Certified Scrum Master Training**
  - **Managing Agile Requirements: The Product Owner's Role**
  - **Introduction to Scrum**
- **Agile Analysis**
  - **Effective Use Case Analysis**
  - **Agile Use Case Analysis**
- **Design Patterns**
  - **Design Patterns Explained**
- **Test-Driven Development**
  - **Test-Driven Development: Iterative Development With Refactoring and Unit-Testing**
- **Effective Programming**
  - **Effective Object-Oriented Analysis and Design**
  - **Effective Agile Programming**
- **ASP.NET**
  - **Effective ASP.NET**
  - **Test-Driven ASP.NET**
- **Integrated Custom**
  - **Agile Software Development with Design Patterns**

Business/Corporate Perspective
Lean Software Development

- **Lean Software Development: A Practitioners Course.** Of the many methods that have arisen to improve software development, Lean is emerging as one that is grounded in decades of work understanding how to make processes better. Lean thinking focuses on giving customers what they want, when and where the want it, without a wasted motion or wasted minute. This 2-day course teaches how to apply lean principles such as: Rapid Response, Constant Learning, Built-in Quality, Local Responsibility and Global Optimization to software development by working through real problems with your peers . This course is based on the work of Mary and Tom Poppendieck.

- **Lean Software For Managers** What do PatientKeeper, a hospital data management system, and Zara, a high fashion clothing chain, have in common with Dell Computer and Toyota Motor Corporation? All four companies are overwhelming their competition with a constant flood of new products that seem to be exactly what customers want, even as they set the standard for quality and value. How do they do it? To these companies, Lean Thinking is a way of life: Rapid Response, Constant Learning, Built-in Quality, Local Responsibility and Global Optimization are part of the culture.

Project Management Perspective
Agile/Scrum

- **Certified ScrumMaster Training.** Agile project management is as radically different from traditional project management as agile processes are different from traditional methodologies. One of the most popular agile methods is called Scrum, and this course is about managing Scrum projects. Rather than plan, instruct and direct, the agile project manager (called the ScrumMaster) facilitates, coaches and leads. In this course you are certified as a ScrumMaster and learn how to make a development team, a project, or an organization agile. The course consists of lecture, hands-on discussions and exercises, case studies, and examples used to educate you in the way of the ScrumMaster.

- **Managing Agile Requirements: The Product Owner's Role.** Iterative development has proven itself to be successful but, managing business expectations and driving development is still hard. We demonstrate how to strike this balance. Further, we explore the benefits and responsibilities of the Product Owner, arguably the hardest job in software development. The existence of a qualified, empowered, Product Owner with analytic support is one of the major success factors for processes.

- **Introduction to Scrum.** Learn why Scrum is an easy to apply agile software development method that can apply to large and small organizations alike. All students who successfully complete this course become Certified Scrum Masters.

Technical Perspective
Agile Analysis

- **Effective Use Case Analysis.** Use Cases are the best tool known for capturing, documenting, and validating the functional requirements for a system. Unfortunately, trying to capture a system's use cases all at once often leads to a severe case of "analysis paralysis". Our approach to use case development is an incremental one that allows us to develop, refine, and validate use cases as we move to more detailed software requirements. We call this technique the Ever-Unfolding Story, and it is the subject of this course.

- **Agile Use Case Analysis.** Agile software development is based on a simple concept: work on the most important things first, and iteratively look for what is 'now' the most important thing. By doing this a development team is able to adapt to changes as they occur, and maximize effectiveness of the development effort. We use the term "Agile Use Cases" to refer to an incremental technique for iteratively developing Use Cases and refining them into software requirements using the Ever-Unfolding Story. Agile Use Cases provide for frequent validation of the requirements, thus supporting agile development.

These courses are similar in content, but with different focus, the first is for larger, more traditional, projects; while the second is designed to support agile developments. Each of these 3-day courses consists of lecture and hands-on exercises, and the use case portion is largely based on Alistair Cockburn's book "Writing Effective Use Cases" - winner of the Jolt Productivity Award for 2001.

Technical Perspective
Design Patterns, Test-Driven Development

- **Design Patterns Explained: A New Perspective on Object-Oriented Design.** This 3-day course goes beyond merely teaching several design patterns. It also teaches the principles and strategies that make design patterns good designs. This enables students to use these advanced design techniques in their problems whether design patterns are even present. After teaching several patterns and the principles underneath them, the course goes further by showing how patterns can work together to create robust, flexible, maintainable designs.

- **Test-Driven Development: Iterative Development With Refactoring and Unit-Testing.** 3 days. The practice of Agile Software Development requires, among other things, a high degree of flexibility in the coding process. As we get feedback from clients, stakeholders, and end users, we want to be able to evolve our design and functionality to meet their needs and expectations. This implies an incremental process, with frequent (almost constant) change to the code we're working on. Refactoring, the discipline of changing code without harming it, is an essential technique to enable this process. Unit testing, which ensures that a given change has not caused an unforeseen ripple effect in the system, is another.

Technical Perspective
Effective Programming

- **Effective Object-Oriented Analysis and Design.** This 5-day course is taught in C++, C#, and/or Java. C++, C#, and Java are powerful programming languages. Their true value emerges only if they are applied with good object-oriented practices. Without these, they can be difficult languages to maintain and extend. Although many people understand the basics of object-oriented programming (polymorphism, interfaces, encapsulation, ...) they don't understand how to use these effectively. This course starts with a better way to do analysis than is traditionally taught. It follows up with 14 practices that both enable better object-oriented coding as well as create a better understanding of object-oriented design. These practices come from both design patterns and new agile coding methods. (This course is about 40% lab)

- **Effective Agile Programming.** Requirements change. This is the new (or not so new) mantra of software development. Changing requirements mandate changeable code. This 3-day course deals with how we write changeable code. It covers issues of code quality, core coding practices, basic object-orientation, refactoring and unit-testing. It then discusses the issue of how to deal with existing legacy code.

Language Support
ASP.NET Training

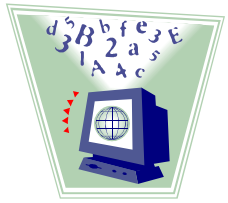
- **Effective ASP.NET.** Microsoft's ASP.NET is a powerful new technology for developing web applications. Its very power of being a simple tool for building robust applications also gives it the tendency to generate maintenance nightmares, if not used appropriately. The problems begin when the application grows beyond a few simple pages. Solid design techniques must be used to keep such applications flexible and maintainable. We take the time to clearly reveal how web applications work, and how they differ from traditional GUI or console-based applications.

- **Test-Driven ASP.NET.** Test-Driven Development (TDD) is a powerful tool for combining software design, testing, and coding to increase reliability and productivity. With ASP.NET, however, Microsoft has created a tool that doesn't easily lend itself to test-driven development. Furthermore, the ease of creating ASP and ASP.NET applications has led to established applications that don't have any tests, making changes difficult and risky. Savvy developers have been looking for ways to apply test-driven development to both new and existing code so they can make changes quickly and safely. In this course, we teach you how to use NUnitAsp to perform test-driven ASP.NET. We show you "best practices" for performing test-driven development of new ASP.NET and then dive into the challenges and solutions of adding tests to existing code. We provide guidance in test-driven development, NUnitAsp, and throw in lots of pithy comments derived from years of experience with architecting web applications.

Customized
Integrated

- **Agile Software Development with Design Patterns.** This 5-day course analyzes what it means to be an agile project, and provides a number of best practices that enhance agility (focusing on XP). After teaching several patterns and the principles underneath them, the course goes further by showing how patterns can work together with agile development development strategies to create robust, flexible, maintainable designs.

Bibliography



Best place to buy books: www.bestbookbuys.com

Full bibliography at

http://www.netobjectives.com/wikis/netobjectives/owbase/ow.asp?RecommendedBooks

Bibliography – Table of Contents

- Our works
- Patterns – core
- Patterns – Architectural, Network, Multithreaded
- The UML and Use Cases
- Miscellaneous
- Java
- C++
- Agile – Management Oriented
- Agile – Technically Oriented
- Resources For Unit Testing in the J2EE
- On-Line Books

Bibliography - Us

- We write many original articles as well as offer several seminars in an on-line, on-demand format. Please go to our own library for these resources as well as further recommendations.
  - http://www.netobjectives.com/resources/rs_library.htm

Bibliography – Patterns - Core

- Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helms, Johnson, Vlissides (still the best reference book on patterns)
- Design Patterns Explained: A New Perspective on Object-Oriented Design, Shalloway, Trott
- Design Patterns Java Workbook, Metsker (good for Java implementation)
- Head First Design Patterns, Freeman, Freeman, Bates, Sierra. A good intro to patterns. Covers more patterns but fewer concepts than Design Patterns Explained, but still a great introduction.
- Timeless Way of Building, Alexander (a classic. The only book in my top ten personal and top ten business). Fun to read, gives great insights.

Bibliography – Patterns – Architectural, Network, Multithreaded

- Core J2EE Patterns: Best Practices and Design Strategies, Alur, Crupi, Malks. This book is good for anyone writing distributed network applications as many of the issues deal with network performance on a conceptual level. However, if you are doing J2EE work, this is an essential read.
- Patterns Of Enterprise Application Architecture, Fowler, et. al.
- Pattern Oriented Software Architecture Vol 2: Patterns for Concurrent and Networked Objects, Schmidt, Stal, Rohnert, Buschmann, (great book for multithreaded, distributed patterns)

Bibliography - The UML and Use Cases

- Writing Effective Use-Cases, Cockburn. The best book on use-cases by far.
- Patterns for Effective Use Cases, Adolph, Bramble, et. al. with contributions from our own Dr. Dan Rawsthorne
- UML Distilled, 3rd Ed., Fowler (easiest and best way to learn what the UML is)

Bibliography - Miscellaneous

- Common Knowledge: How Companies Thrive by Sharing What They Know, Dixon. Good book on communities of practice and knowledge management.
- (The) CRC Card Book , Bellin, Simone (CRC cards are a great way to verify initial designs)
- Multi-Paradigm Design for C++, Coplien (excellent description of commonality/variability analysis -- I even recommend the first part for Java programmers. See http://www.netobjectives.com/download/CoplienThesis.pdf for the book's equivalent on the web)

Bibliography - Java

- Effective Java, Bloch. An excellent book. Read a chapter at a time to best enjoy and absorb it.
- Java Design, Coad (best book on how to do OO in Java by far)
- Thinking in Java (3rd Ed), Eckel. Great book to learn Java and object-oriented programming from.

Bibliography – C++

- Accelerated C++, Koenig, Moo. The best book to learn C++ from. Does not require C knowledge, but that doesn't slow you down. An unusual teaching style that works. We use this in our C++ course.
- C++ Coding Standards: 101 Rules, Guidelines, and Best Practices (C++ in Depth Series), Sutter, Alexandrescu. Great, practical book.
- C++ Common Knowledge: Essential Intermediate Programming, Dewhurst. Becoming my favorite of the "Effective C++" type books.
- Effective C++, Meyers. Also used in our C++ course. Teaches you both to avoid pitfalls and take advantage of strengths.
- Thinking in C++ (2rd Ed), Eckel. Great book to learn C++ and object-oriented programming from. A little more difficult than Accelerated C++ but goes deeper into OO.
- Advanced Programming in the UNIX(R) Environment, Stevens. If you are on UNIX/LINUX using C/C++ this is a must.

Bibliography – Lean Thinking

- Agile Management For Software Engineering. Anderson. Brilliant, though at times spotty on accuracy. Still a great read for ideas.
- Lean Software Development: An Agile Toolkit for Software Development Managers. Poppendieck, Poppendieck. ***An absolute must read.*** Lean is time-proven and management friendly. Mary and Tom have done a brilliant job here.
- Lean Thinking : Banish Waste and Create Wealth in Your Corporation. Womack, Jones
- The Goal. Goldratt, Cox. The first and still best book on the Theory of Constraints.
- Toyota Production System: Beyond Large-Scale Production, Ohno. Brilliant book that proposes a few key concepts that really apply everywhere.

Bibliography – Agile Management Oriented

- Agile Software Development. Cockburn. Great insights on teams.
- Agile Software Development With Scrum. Schwaber, Beedle. ***An absolute must read.***
- Software by Numbers: Low-Risk, High-Return Development. Denne, Cleland-Huang
- Waltzing With Bears: Managing Risk on Software Projects. DeMarco and Lister. Great book on how to measure and handle risk.

Bibliography – Agile XP Centric

- Extreme Programming Explained: Embrace Change, Beck. Although somewhat dated now, still chock full of brilliant ideas.
- Extreme Programming Installed, Jeffries. Good practical information on XP.

Bibliography – Agile Technically Oriented

- Agile Software Development, Principles, Patterns, and Practices, Martin.
- Emergent Design: The Evolutionary Nature of Professional Software Development, Bain. 2005. For previews: http://www.netobjectives.com/emergentdesign/ed_toc.htm
- Java Tools for Extreme Programming: Mastering Open Source Tools Including Ant, JUnit, and Cactus, Hightower, Lesiecki
- Refactoring, Fowler (most in depth book on this old practice)
- Test Driven Development: By Example, Beck. Not as great as we were hoping, but very useful.

Bibliography – Acceptance Testing Tools

- Fit for Developing Software: Framework for Integrated Tests, Mugridge, Cunningham. The book on Fit.
- Fit – Framework for Integrated Tests
  - http://fit.c2.com/
- Fitnesse – Team Use of Fit
  - It is a self-contained, fully integrated, wiki and acceptance testing framework based upon FIT.
  - Fitnesse.org

Bibliography – Resources For Unit Testing in the J2EE

- JUnitEE. JUnitEE is a simple extension to JUnit which allows standard test cases to be run from within a J2EE application server. It is composed primarily of a servlet which outputs the test results as html. www.junitee.org
- EJB Unit Test page on a popular wiki. The purpose of this page is to discuss and possibly define some guidelines for doing unit tests on Enterprise Java Beans. www.c2.com/cgi/wiki?EjbUnitTest
- Test infect your Enterprise JavaBeans. Learn how to test your J2EE components live and in the wild. www.javaworld.com/javaworld/jw-05-2000/jw-0526-testinfect.html
- Cactus. You might also want to look at this. It's significantly more complicated than JUnitEE, but it provides the http servlet objects to your test case. jakarta.apache.org/cactus
- JTest. A presumably good, but expensive product from Parasoft. parasoft.com/jsp/home.jsp

On-Line Books

- Many books are available on-line at:

http://www.netobjectives.com/wikis/netobjectives/owbase/ow.asp?RecommendedBooks

Specific Articles of Great Interest

- Test Infected: Programmers Love Writing Tests

A great article on doing test driven development with Junit by Gamma and Beck - the authors of JUnit.
http://members.pingnet.ch/gamma/junit.htm

Famous Sayings

- The check is in the mail.
- I'll respect you in the morning.
- We've got all the requirements.
- That'll never happen.
- That always happens.
- I'll write the tests later.
- I can write bad code because I know refactoring….
- Yes, I'm sure.
- We'll integrate it in later.
- I'll fix it later.

Notes