IBM Rational Software Conference 2009

Monojit Basu
Product Manager, Rational Change and Configuration Management
monobasu@in.ibm.com

Rational. software

Go to IBM

Agenda

- Introduction
- Typical Development Environments
- Understanding Agile
- Agile Methods - Scrum, Requirements, Modeling, TDD, CI
- Game Plan to adopt Agile Methods

Typical Development Environments

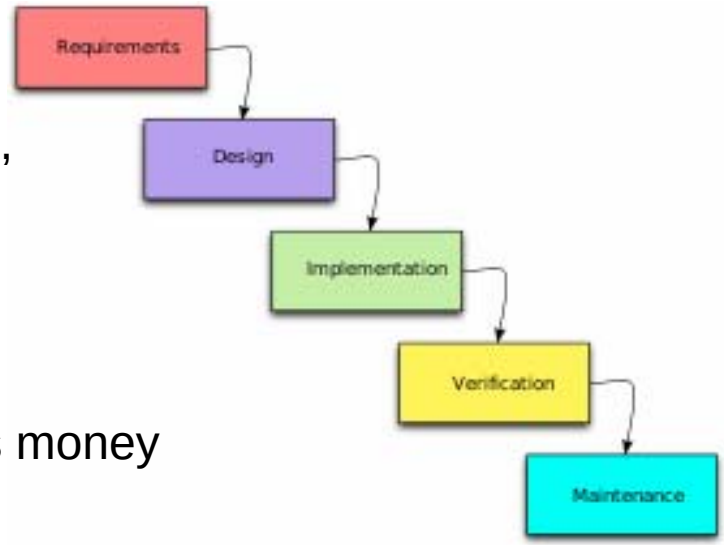
- Which of these scenarios apply to your company?
  - A large corporation of employees with highly visible projects that have trouble delivering on time or adapting to change because the existing process is elongated
  - A company that has multiple development sites that are geographically dispersed between hemispheres or across the continent
  - A small or medium sized company of highly skilled workers unable to demonstrate value and quick turnaround in a repeatable fashion
- Many companies fall into one or more of these scenarios looking to for a way to help with the pain
- Agile techniques and practices are gaining more and more attention in the software community
  - What's not clear are the differences between the techniques
  - Which techniques should be adopted for your organization or project?
  - Then there's the fear that you may lose productivity

Typical Development Environments

- Software development has evolved and changed drastically in the last 20 years
  - Technologies have evolved
  - Framework have been developed
  - Complex Tools created to ease the burden of developing software
- Software development life should be easier by now right?
  - Truth is, software development has gotten heavily burdensome and overly complicated
  - What used to be quick in today's environment takes much longer to accomplish
  - There just seems to be too many steps getting in the way of the primary goal to develop software

Typical Development Environments - Waterfall

- In order to understand how or what to adopt, it's important to understand what Agile really means and how it differs from traditional development
- To set the stage, consider the Waterfall based process:



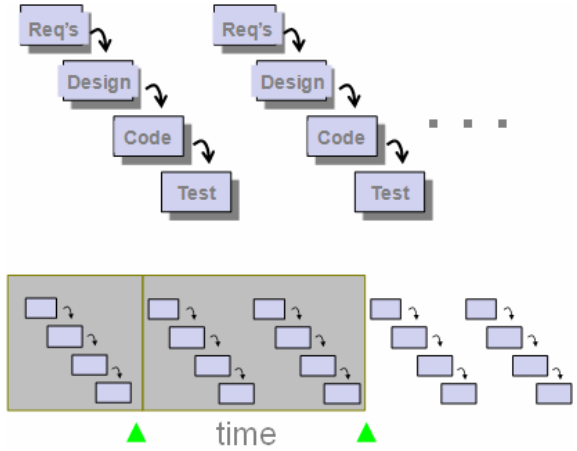
- Pros:
  - Simple, Well-defined milestones, Disciplined, Ordered
  - Still used by the majority of large companies
- Cons:
  - Risk is delayed until the very end. This costs money if things go wrong.
  - What if the architecture won't work?
  - What if System Testing identifies a major bug – or worse, missed requirements?

Typical Development Environments - Waterfall

- Waterfall based processes demand thick documentation with a focus on many physical artifacts (documents) that ended up becoming a series of forests cut down
  - Why? It's a hand-off – sometimes between groups
  - The single "Requirements Document" consisting of many, many pages handed over to developers, testers, and business stakeholders
    - These are filled with declarative statements such as "The System shall …."
  - In many cases, one document artifact gets copied or translated into other similar documents
    - Requirements Specification to Technical Specifications to Design Specifications
  - All the while the notion of what the system is really supposed to do or how someone would use that system becomes harder to see clearly amid all of the pages of documentation
  - Maintaining a system like this or adding new capabilities becomes more and more difficult

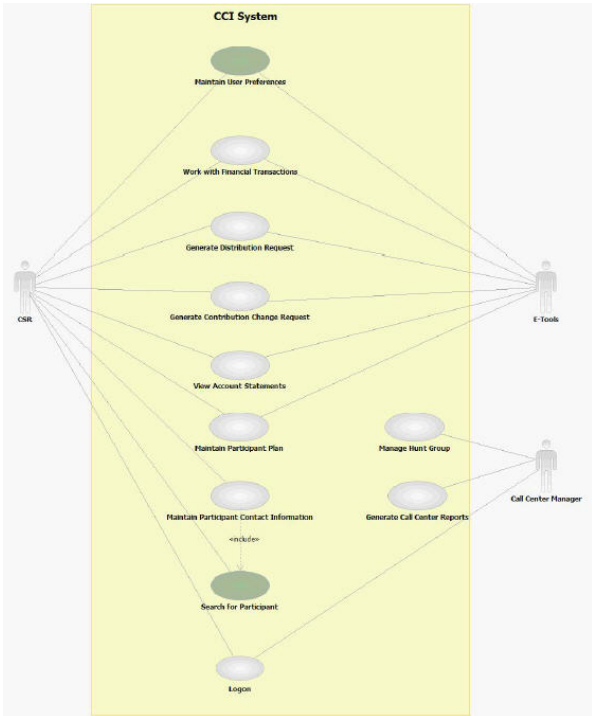
Typical Development Environments - Iterative

- Then along came iterative based development
  - It too was well defined
  - Risks were resolved earlier by developing incremental behavior of the system and integrating sooner
  - Unlike Waterfall – which had a single release – functionality could be deployed earlier in the lifecycle in a series of releases
  - RUP, Spiral, MSF, OpenUP are examples of iterative processes
  - Software created in time-boxed iterations
  - Each revision incrementally builds upon the last working and tested software



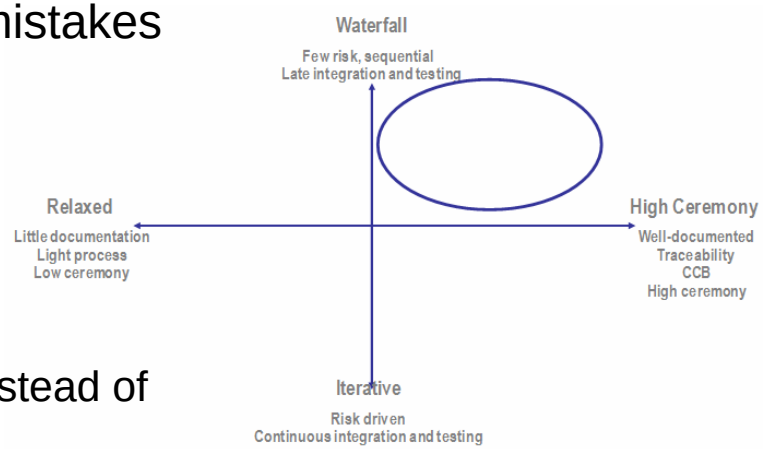
Typical Development Environments - Iterative

- Iterative development processes like the RUP take a different approach
  - Use Cases used to capture functional requirements of a system organized in a sequence of steps that yield some observable result to the external Actor requesting it
  - Use Cases focus on how someone or something uses a system – the usage model
  - Actors are used to abstract someone or something outside the boundary of the system being developed
- Benefits:
  - Easier to understand since they are written in the language of the customer
  - Puts requirements into the appropriate context of system responsibility
  - Facilitates re-use across iterations



Typical Development Environments - Iterative

- However, when adopting an iterative process, mistakes can happen
  - RUP for example is a Framework – not necessarily a specific process
    - There are over 2500+ web pages to sift through
  - Might be morphed into an artifact based process instead of one that delivers value and resolves risk
  - Focus at first addresses technical Architectural Risks rather than delivering Business Value to the customer.
    - “It’s a great design. Too bad we didn’t have it 3 months ago when we really needed that front end piece.”



Understanding Agile

- Agile methods are not Cowboy programming as depicted by some in the media
- In fact, some of the key practices of Agile methods include:
  - Iterative and incremental development
  - Two levels of planning
  - Frequent releases
  - Self-organizing teams
  - Just-in-time requirements specification
  - Concurrent testing (developer & system)
  - Continuous Integration
  - Refactoring
  - Unit testing & Test Driven Development (TDD)
  - Retrospectives

Understanding Agile

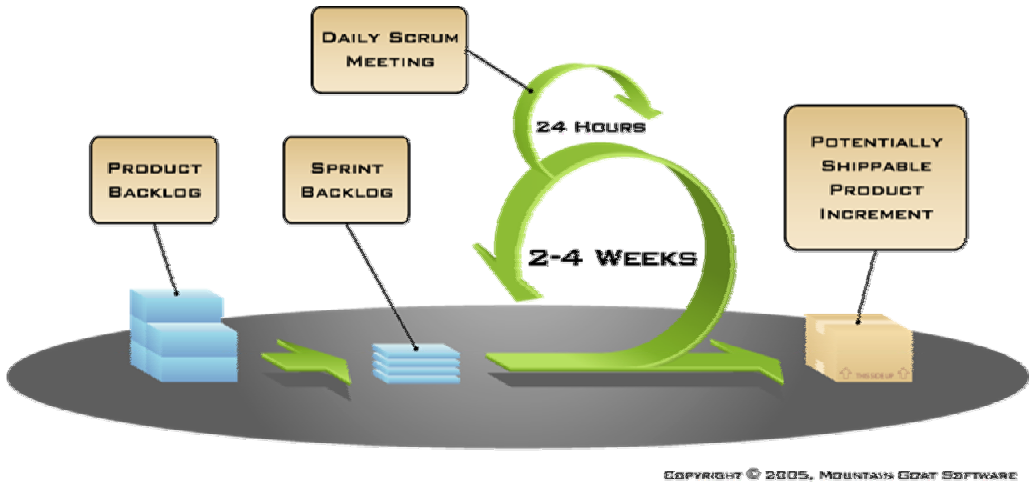
- There is more than one method in the Agile camp. Each has its own strengths and areas of focus
- All Agile methods typically conform to the same basic Principles:
  - Lightweight, minimalist, low-ceremony
  - Highly collaborative: People over process and tools
  - Value-driven: Frequently provide value to end users
  - Embrace Change: Adapt instead of Predict
  - Feedback: Promote extremely tight response loop
  - Build Quality In: Adopt a “stop the line” mentality

Understanding Agile

- All Agile methods are adaptive in nature
  - The cadence of agility is to expect change for incremental release and to react to those changes
  - This type of feedback allows the team to respond to change as opposed to following a strict plan
  - Customers see results sooner and therefore have more input for what to do next

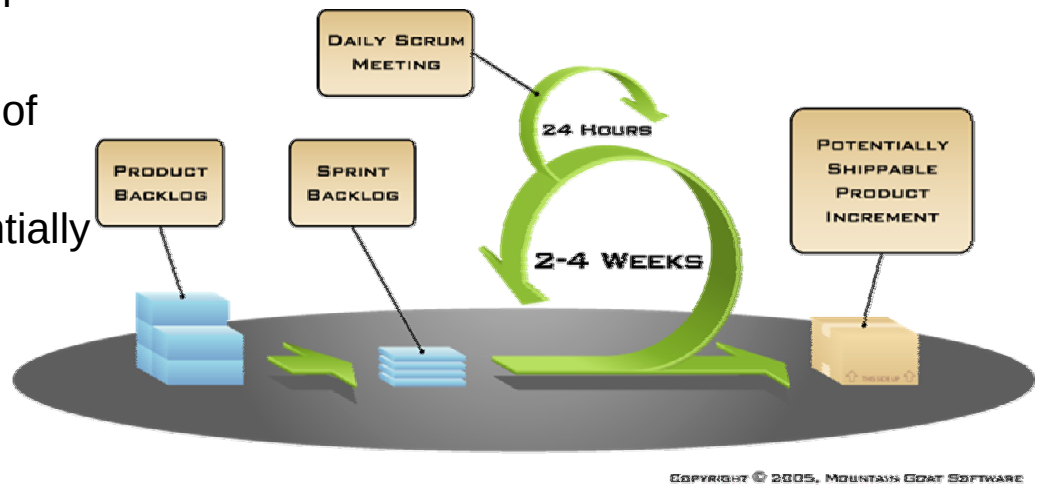
Agile Methods – Scrum Overview

- Every project starts off with planning but many of today's plans are educated guesses that are padded for safety
- Scrum is an Agile Project Management Framework based upon empirical evidence
  - Scrum does not prescribe specific software engineering technique such as requirements gathering, testing, or development



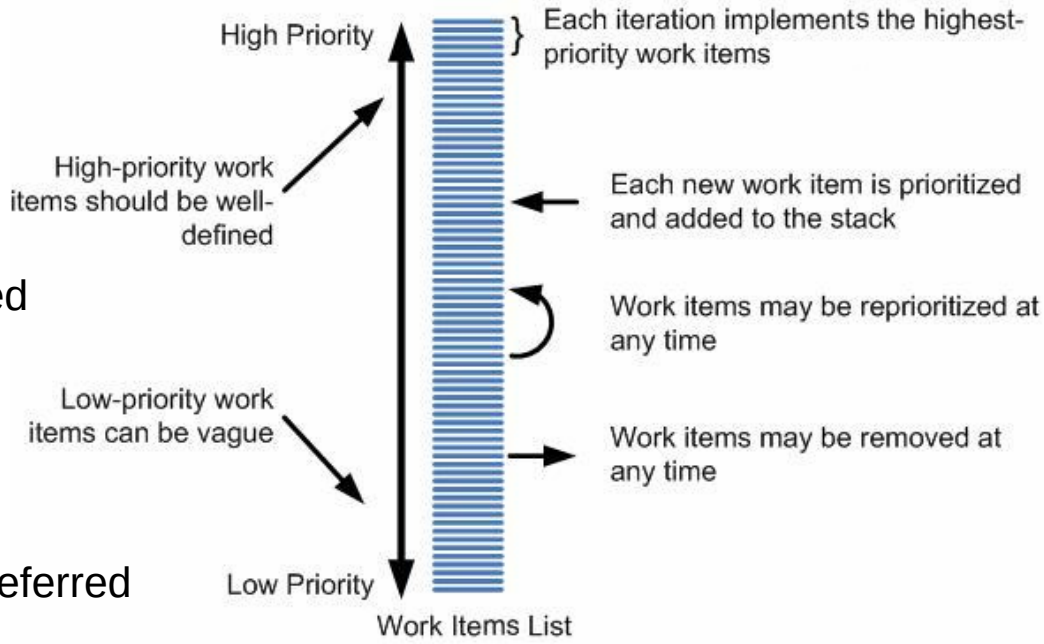
Agile Methods – Scrum Overview

- Scrum uses two levels of planning:
  - Requirements, Features, Defects, etc. are all organized into virtual work item lists called Backlogs
  - The Product Backlog is managed by a Product Owner who ranks each item based upon the value it brings
  - Development is then done in a series of time boxed iterations called Sprints
  - At the end of a Sprint, you have potentially shippable code



Agile Methods – Scrum Overview

- Product Backlog
  - Estimated by the Team using relative estimation measurements called “Points”
  - Points are estimated per Backlog based upon their size and difficulty relative to other Backlog items
  - You don’t need all of the requirements detailed before you can begin
  - Commitments to the requirements is deferred until the Sprint planning begins
  - Requirements are detailed just-in-time for them to be worked



Agile Methods – Scrum Overview

- Sprint Backlog
  - This is a detailed iteration plan that is created by the team by pulling off items from the Product Backlog
  - Each product backlog item is then broken down into individual tasks necessary to complete the work
  - Each task's work effort in is estimated collaboratively in hrs by the Team
    - Not to exceed 16 hrs
  - Team members then self-assign the tasks they want to do
  - Estimated remaining hrs for the task is updated by the individual team member daily

As an Online Shopper, I want to be able to browse for products

Develop middleware	- 8 hrs
Create UI	- 4 hrs
Code tests	- 4 hrs
Code xxx Class	- 6 hrs
Refactor	- 4 hrs

Agile Methods – Scrum Overview

- Daily Scrum
  - Each day, the team comes together to answer 3 basic questions:
    - What did I do yesterday?
    - What am I going to do today?
    - What's in my way?
  - This is a short (15 minutes) meeting to give status to the rest of the team
    - It's not to solve problems but rather to foster communication
    - It indicates commitment of the work items
  - Anyone can attend but only the Team, Scrum Master, and Product Owner can contribute

Agile Methods – Scrum Overview

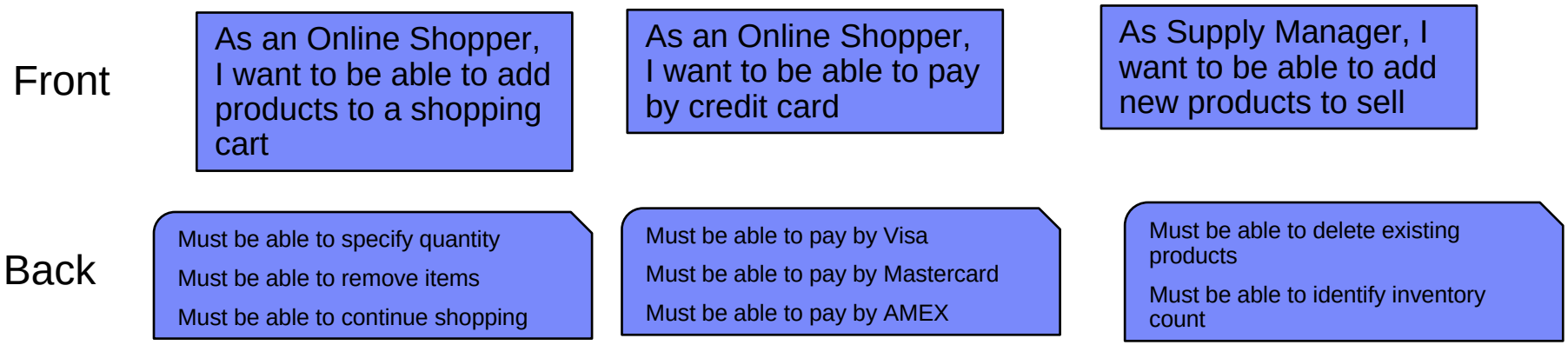
- Sprint Review
  - At the end of the Sprint the team comes together to show their accomplishments during the Sprint
  - Typically this is done as a live demonstration and the Product Owner can accept or reject the changes
    - Rejections or defects are added back onto the Product Backlog
    - Accepted items' relative point estimates are accumulated for the Sprint into a total known as "Velocity" – which measures how much work a team can do in a Sprint
  - All team members participate in the demo
    - Does not require days of preparation. In fact it should be 2 hrs or less
  - Like the Daily Scrum, anyone can attend
- Sprint Retrospective
  - After the Sprint review, the team comes together to discuss what went well and what didn't go so well
  - The goal is to self-improve

Agile Methods – Agile Requirements Overview

- User Stories are an Agile Requirements technique to describe discrete functionality that benefits the customer
- They encourage two agile principles:
  - Shifts the focus to conversation over the written word
  - Defers commitment until it is needed to build
- Perfect size for planning purposes
- Understood by the customer and the developer
- Excellent for enhancing an existing application
- They are smaller than a use case

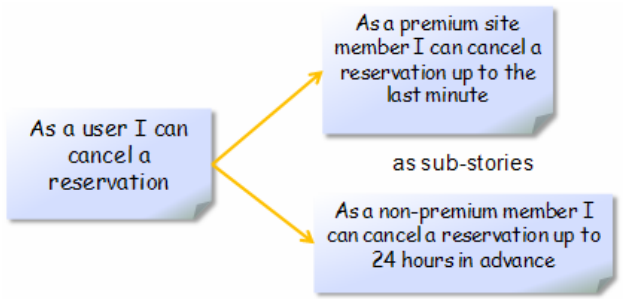
Agile Methods – Agile Requirements Overview

- Each User Story is composed of three main parts:
  - A Short description no more than a sentence or two
  - Acceptance tests to confirm the story is complete
  - Verbal conversations that elaborate the story for understanding
- Typically written by the Customer or Product Owner
- They can be captured on a 5 x 9 index card or electronically captured in an agile tool such as RTC



Agile Methods – Agile Requirements Overview

- Can be broken up into sub-stories if more detail is needed
- Each story should be:
  - Independent
  - Negotiable
  - Valuable
  - Estimatable
  - Small
  - Testable
- Organized in terms of: MoSCoW Rules
  - Must – Fundamental to the system
  - Should – important by not required
  - Could – could be left out
  - Won't – not necessary this time around so it can wait



Agile Methods – Agile Requirements Overview

- Using the MoSCoW rules helps the Customer and Product Owner prioritize the Product Backlog
- Other techniques involve specifying whether this story is a Market Differentiator or not
  - Those that may have more weight
  - Note that a Market Differentiator doesn't have to be a Must category
- The Product Backlog can be re-prioritized at any point in time
  - This gives the freedom to the Customer to control the feature efforts based on value to the business – what a concept

Agile Methods – Agile Modeling

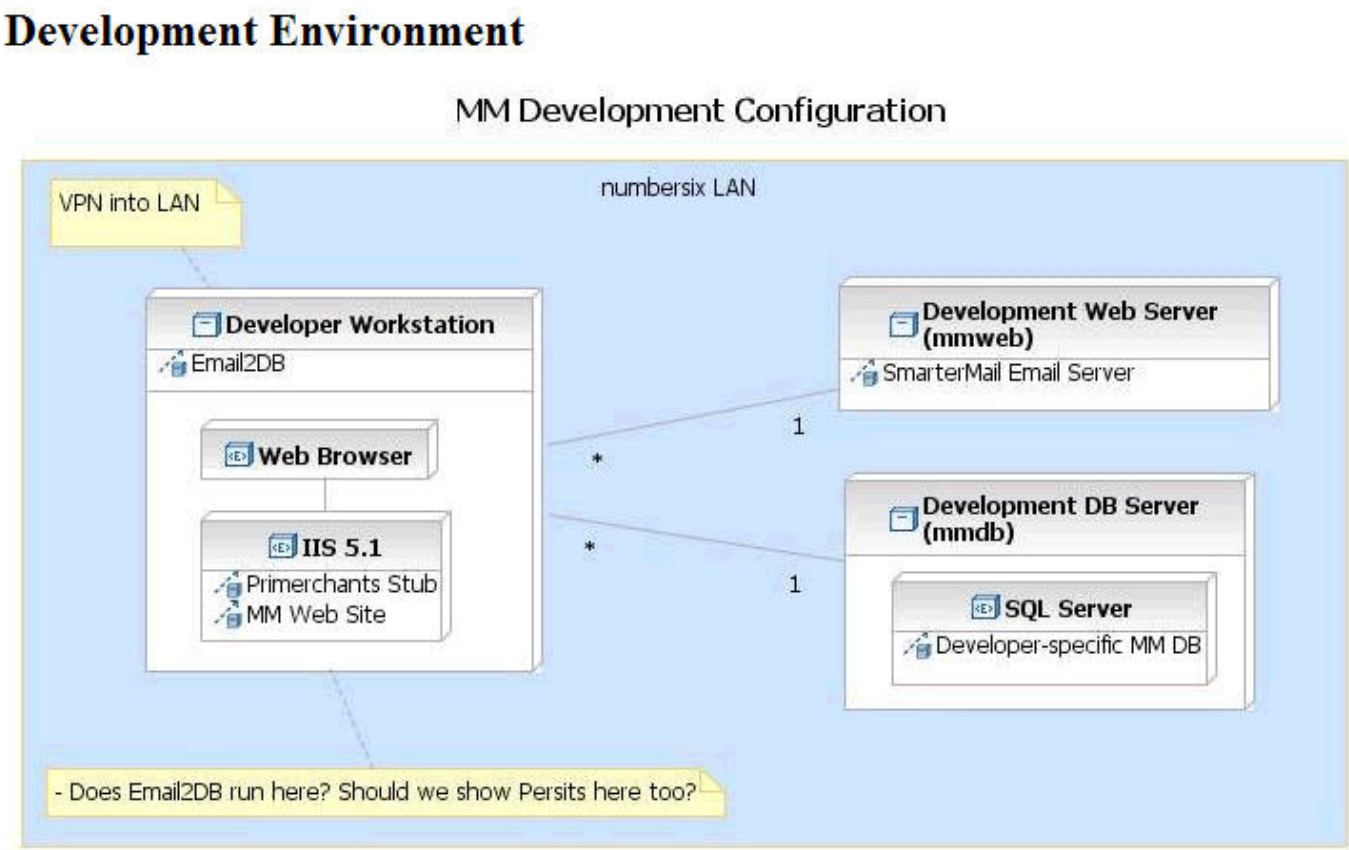
- Agile Modeling is an agile technique that is applied across multiple disciplines
  - The common thread in the above disciplines is that each of them have a common activity at its core – that is the need to model
  - Models are an abstraction of a problem at a certain perspective using a common notation
- RUP is very prescriptive about Models
  - Business Use Case models to system Use Case models
  - Domain models to Analysis models to Design models to Implementation models
- Tools perpetuate this notion through their support of notations such as UML 1.x & 2.x, BPMN, BPEL, etc.
- RSA for example, not only supports the 13 UML 2.x diagram types but also adds additional diagrams like Topic and Browse diagrams. Do we really need all of them for a project?
- The short answer is: NO!!

Agile Methods – Agile Modeling

- There are really two kinds of models:
  - Formal models built with complex modeling tools
  - Informal models built on a whiteboard, digital picture, or simple modeling tools like Visio
- Each model serves different audiences and a project usually has more than one model
- Models help teams think through a problem
  - a picture really does eliminate misunderstanding of the spoken word
- All code ever developed came from one or more models
  - Either a formal set of diagrams, a whiteboard concept, or formed in the developer's mind before writing the code
- Modeling can be extended into supporting practices such as MDD and MDA

Agile Methods – Agile Modeling

- Model when it's faster than writing text
- Original text was too confusing. It needed a model to be explained



Agile Methods – Agile Modeling

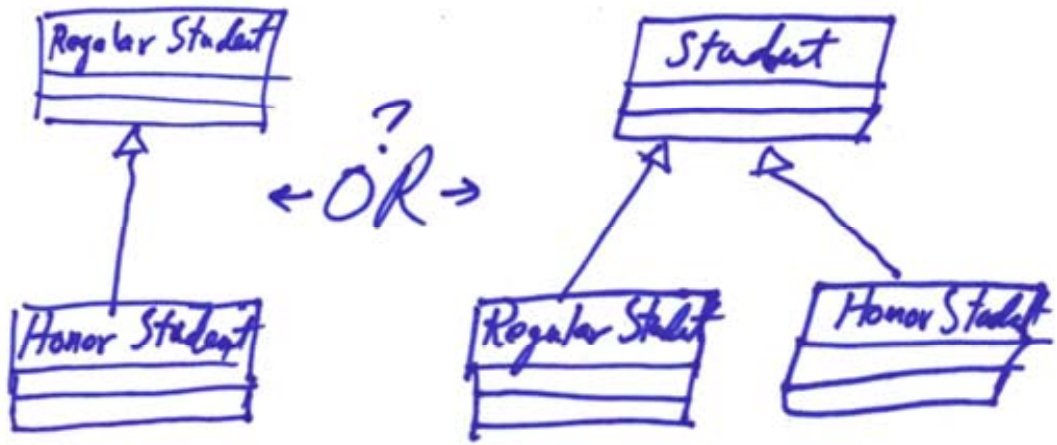
- Adoption of Agile Modeling techniques means you only model what you can't do without
  - When it actually hurts your project not to have that model
- Always model with a purpose in mind – not because it is "part of the process"
  - If it servers no purpose – don't do it
- Only model enough to support the effort
- Don't model for perfection. It just needs to be good enough to:
  - Get your point across
  - Meet the Intended Audience expectations
  - Clear and Detailed when necessary
  - Reliable
  - Simple
  - Holds value
- There is a cost to maintaining models

Agile Methods – Agile Modeling

- Adopting Agile Modeling takes a certain approach.
  - Sometimes it really isn't necessary to model at all
    - Do you need a model when all you are doing is adding a method to a class?
  - Agile models that are created and actually kept are a small percentage of the entire effort
    - Models that need to last longer than the project or for teams geographically distributed are candidates to keep and maintain
    - When the cost of NOT having the model outweighs the cost of rebuilding it which slows down development
    - Example keepers are Architectural Models, high visibility Activity steps, or complex Business Rules
  - During Agile modeling, many models may be created but a large majority are transient and are tossed once they serve their purpose
    - These rarely last past a few iterations
    - Examples would be some context or communication diagrams

Agile Methods – Agile Modeling

- Agile modeling efforts usually don't start in a tool
- Whiteboards, Paper are perfectly acceptable
- The emphasis is to describe visually an idea so that conversation can begin
  - Shortens the development lifecycle by removing unnecessary diagrams
  - Saves money over time by reducing the number of models to maintain
  - Have value and meet a specific purpose for an intended audience



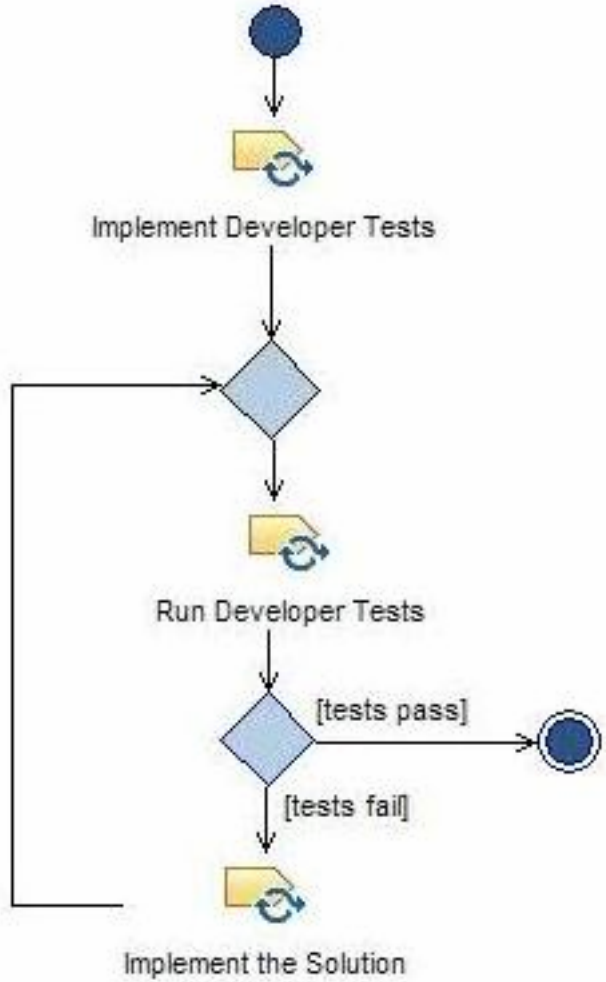
Ali Ali - ATSC

Agile Methods – Test Driven Development

- TDD came from XP but contrary to its name – it's not really geared towards the Testing Discipline in the RUP.
- TDD is a design and programming activity more closely aligned with traditional unit testing than formal testing done by professional testers
- TDD allows developers to organically develop a test suite while building their applications
- In the past, developers would write code and then test it.
  - Unit Tests may had been done manually with little thought to the potential of breaking something else
  - TDD uses automated tests to constrain each functional bit of a program
- TDD is different in that you write the test first and when it fails you write code to pass the test.

Agile Methods – Test Driven Development

- The TDD life cycle repeats until no more features are left to build:
  - Add a test for the new capability feature
    - forces the developer to focus on the requirements before writing the code instead of after
  - Run all tests and see if the new one fails
    - A form of negative testing using all prior tests
  - Write some Code to pass that test
  - Run the Automated tests to see them succeed
  - Refactor Code as necessary
    - Gain knowledge in knowing refactoring will not damage any prior tests



Agile Methods – Test Driven Development

- The are several tools available by choice of programming language that supports TDD such as:
  - JUnit, PHPUnit, TestNG to name a few
- A 2005 study found the using TDD meant writing more tests; developers who wrote more tests tended to be more productive
- TDD can help build software faster by elevating the confidence level that new code will not break existing code
- TDD eventually builds the automated regression Test Harness that can be shared among the team
- TDD can be extended to Test Driven Database Development (TDDD)

Agile Methods – Continuous Integration

- In today's world, not maintaining the code repository with all working parts becomes a hectic task.
  - Developer takes a copy of the code base to write their code
  - Before the developer can check in their code, they must update their code base with that of the repository to catch any changes made by others since (s)he checked out the code
  - The more changes that were made, the more work the developer must do before (s)he can check in their code to the repository
  - Eventually the repository becomes so different from the developer's that it may take more time to integrate the changes than it did to write the new changes
  - This is what's referred to as "Integration Hell"
- Here is where an Agile practice called Continuous Integration (CI) helps out

Agile Methods – Continuous Integration

- Continuous Integration (CI) is simply the practice of integrating components early and often
  - This mitigates the “Integration Hell” some development teams face
- The build frequency is really left open to interpretation and project needs but it is definitely more than once. The term “several times a day” is typically adopted
- The ultimate goal of CI is to reduce that integration time and thus reduce timely rework which ultimately saves costs
- CI Emerged from XP but there are examples of this used by IBM in the 1960’s when they were building OS/360
  - They created builds 4 times each day

Agile Methods – Continuous Integration

- Continuous Integration (CI) basic rules
  - Maintain a repository
    - The system should be buildable from a freshly checked out codebase that is not dependent on other code
    - Rather than creating a new developer branch, it is preferred that changes are integrated directly into the trunk
    - Because of the multiple builds, this is an acceptable practice
- Automate the Build
  - Several tools support this such as:
    - Build automation tools: Make, Ant, Maven, ClearCase
    - Continuous Integration Servers: RTC, BuildForge, CruiseControl, TeamCity, etc.
  - This includes automating the integration in order to deploy to a production like environment
    - Include binaries, web pages, statistics, and possibly documentation
- Make the Build Self-testing
  - Employ TDD techniques to test the code to confirm the behavior

Agile Methods – Continuous Integration

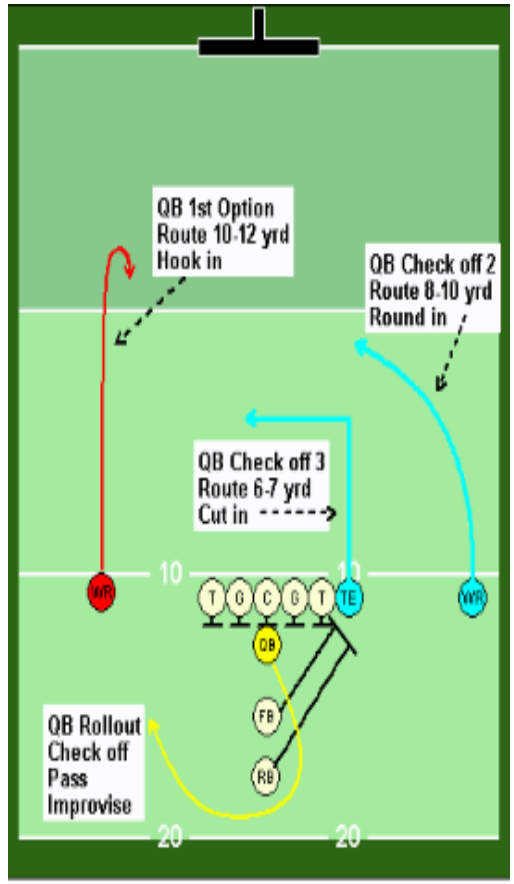
- Commit/Check-in Code Daily
  - This reduces the number of conflicting changes to others instead of committing weeks worth of code at once
  - Each Team member must do this
- Each Check-in to the mainline/trunk should be built
  - This confirms that the integration was successful
  - It can be a manual process or set up as an Automated CI that looks for changes and starts the build
- Keep the Build Fast
  - It can't take hours for the build to complete. You want to know when a problem occurs
- Test the Build in an environment cloned from production
  - This mitigates the errors where a Test environment doesn't match the production environment

Agile Methods – Continuous Integration

- Make the Builds Available
  - That means alerting testers and stakeholders early. This feedback loop reduces the amount of time if the build doesn't meet the requirements
- Publish the Build Results
  - In an ideal world, the results of the build is transparent and made available to the entire team
  - This is helpful to identify when a build breaks and what caused it to break
- Automate the Deployment
  - Creation of a build can be extended by automating the deployment into the target deployment model

What's the Game Plan?

- Create a plan of attack
- Bring in professionals to help assess your needs and plan out the strategy of attack
  - Nothing helps jumpstart an organization like a set of experienced professionals
  - The alternative is a number of whitepapers and books each focused on a particular area
  - Look for a firm with experience who offers a robust set of service offerings that directly relate to your pain points



What's the Game Plan?

- Execute the Plan
- Step 1 - Initial assessment of your needs
- Step 2 – Putting it all together
  - Exposure to more than just one Agile method
  - Formal Training
  - Hands-on Workshops
  - Tool Training to reinforce the concepts learned
  - Ongoing mentoring with real pilot projects

Touchdown!!

Step 1: Assessment

- Every company is different, because of that there cannot be a single cookie-cutter plan
- Your company may already be strong in one area but weaker in another
- Ensure an assessment reflects this and specifies a target set of solutions to address these needs
  - Perhaps planning and delivery is the bottleneck, maybe it's requirements
  - Note that the solutions are usually plural – that's ok, as long as it is part of a cohesive plan
- The assessment should include the existing development team size and location
  - In today's world, development teams are sometimes distributed in multiple buildings, states, or hemispheres
  - Professional firms who claim to be agile, need to be able to embrace this fact

Step 2: Putting it all Together

- Now that we've discussed various Agile methods, let's put the thoughts down on paper and plan how we can do it in around 40 days
- Week 1
  - Conduct assessment and provision the tooling environment
- Week 2
  - Conduct just-in-time training covering:
    - Detailed Agile methods
    - Hands-on Workshops
    - Agile tools (RTC, RRC, RQM)
- Week 3
  - Pilot process and tools with hands-on mentoring from Agile coaches
- Week 4
  - Agile Modeling, TDD & CI Workshops

Step 2: Putting it all Together

- Week 5
  - Mentoring the project with the Agile Coaches
- Week 6
  - Retrospective
  - Plan next Sprint
- The best way to introduce new concepts is not just by the theory of a book
- Combining just enough theory with heavy hands-on workshops helps to reinforce the concepts being described
- Heavy emphasis should be on:
  - Collaboration
  - Planning
  - Instruction
  - Execution


DEMO


Questions