A Convergence of Technologies for Better Software NOW!

Dottie Acton

Lockheed Martin IS&GS

Topics

- Two categories of software errors
- How technologies can help
- Who needs to be involved
- Some experiences
- Questions
- Backup charts with technology descriptions
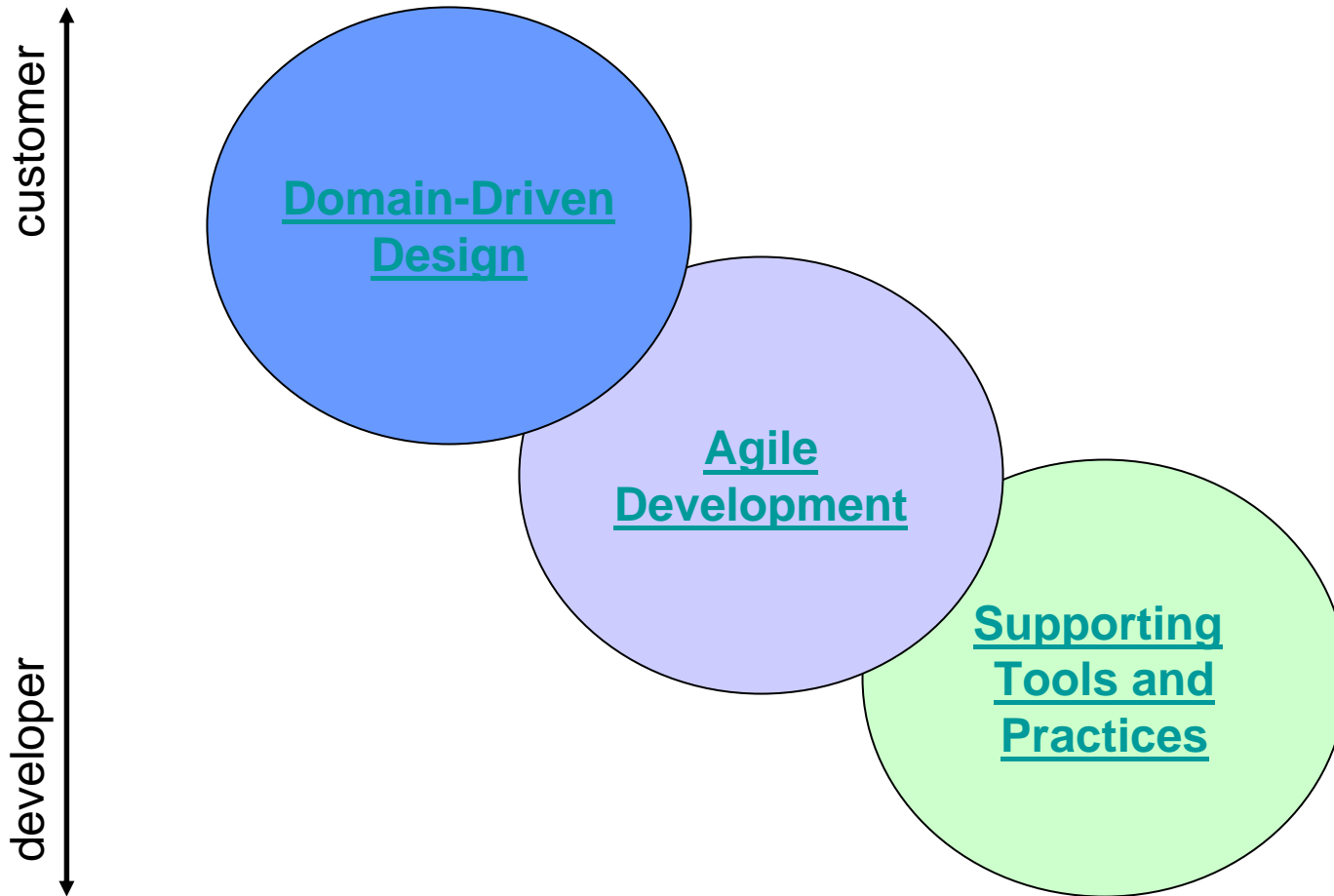
Two Categories of SW Errors

Solving the problem wrong

- Incorrect implementation of requirements (off by one bug, logic errors, etc)
- Bad coding practices that leave security holes
- Interface mismatches
- Delivering too late to be useful
- Un-maintainable code
- Poorly performing software
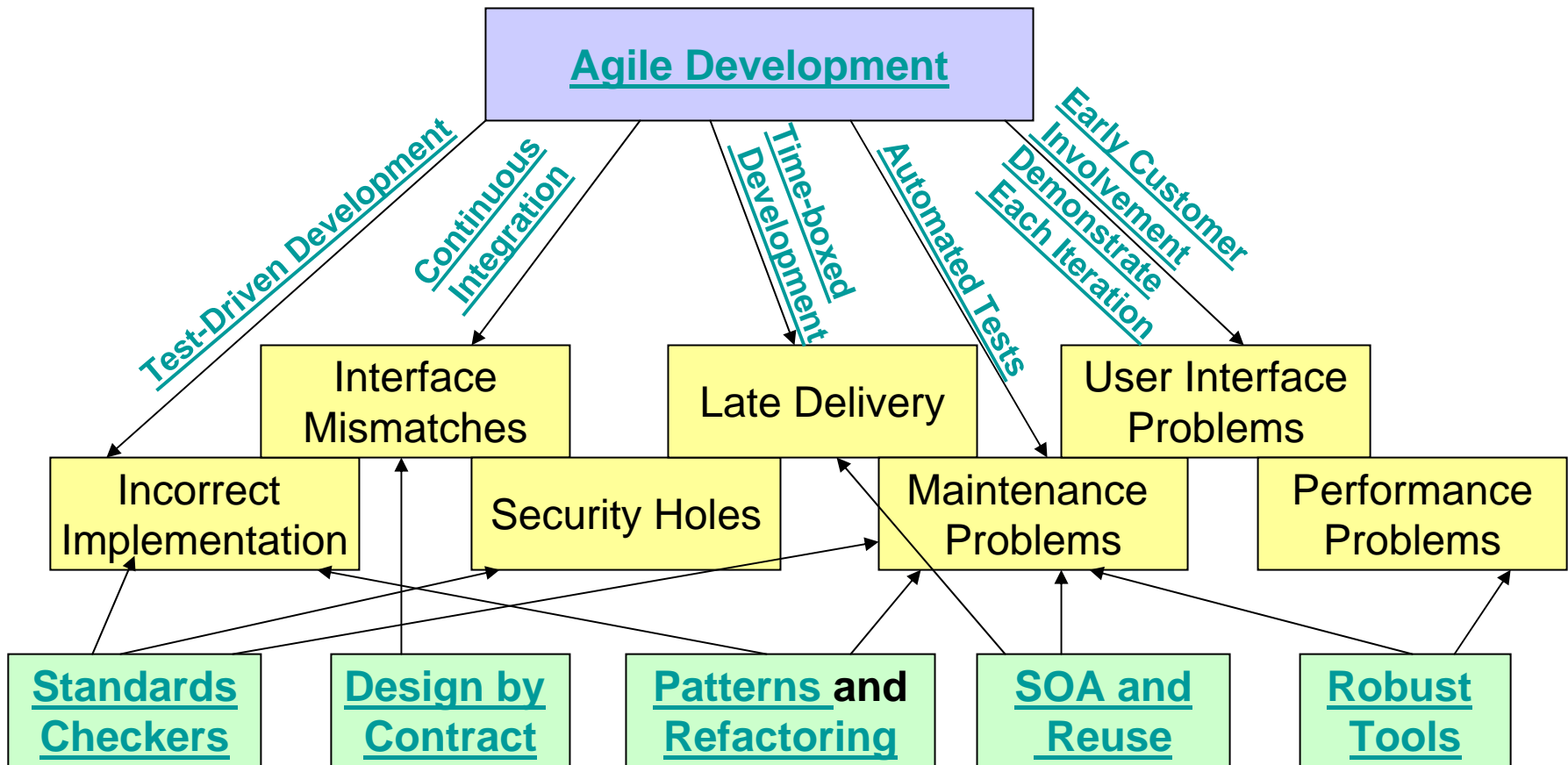- Poor user interfaces

Solving the wrong problem

- Misinterpretation of requirements
- Missing requirements
- Unused capabilities
- Obsolete requirements
- Failing to recognize the existence of 'wicked' problems (the solution changes the nature of the problem)
- Focusing on generic or supporting domains rather than on the core domain

3

What are the Technologies?

customer

developer

Domain-Driven Design

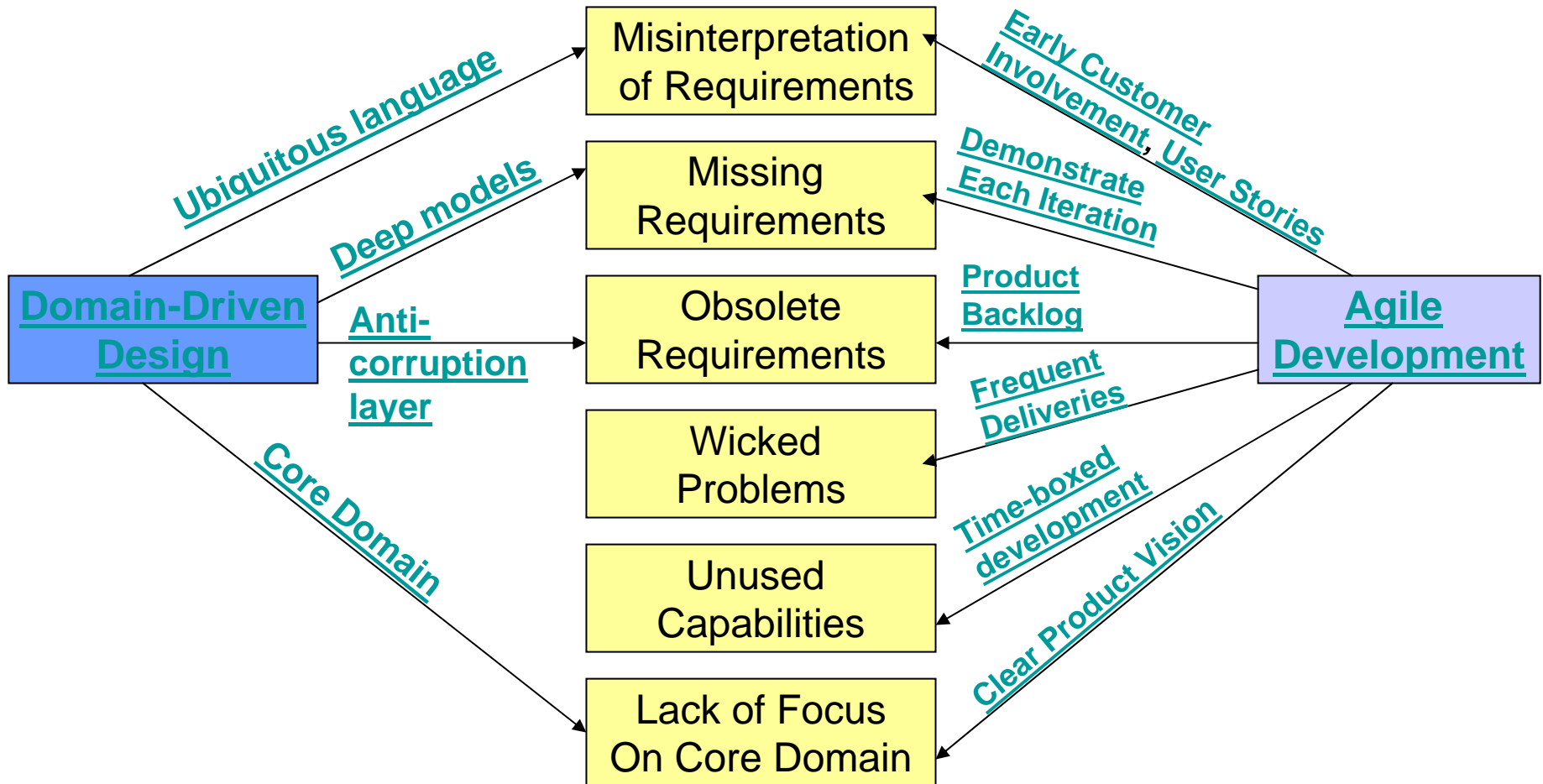**Agile Development**

Supporting Tools and Practices

The Good News

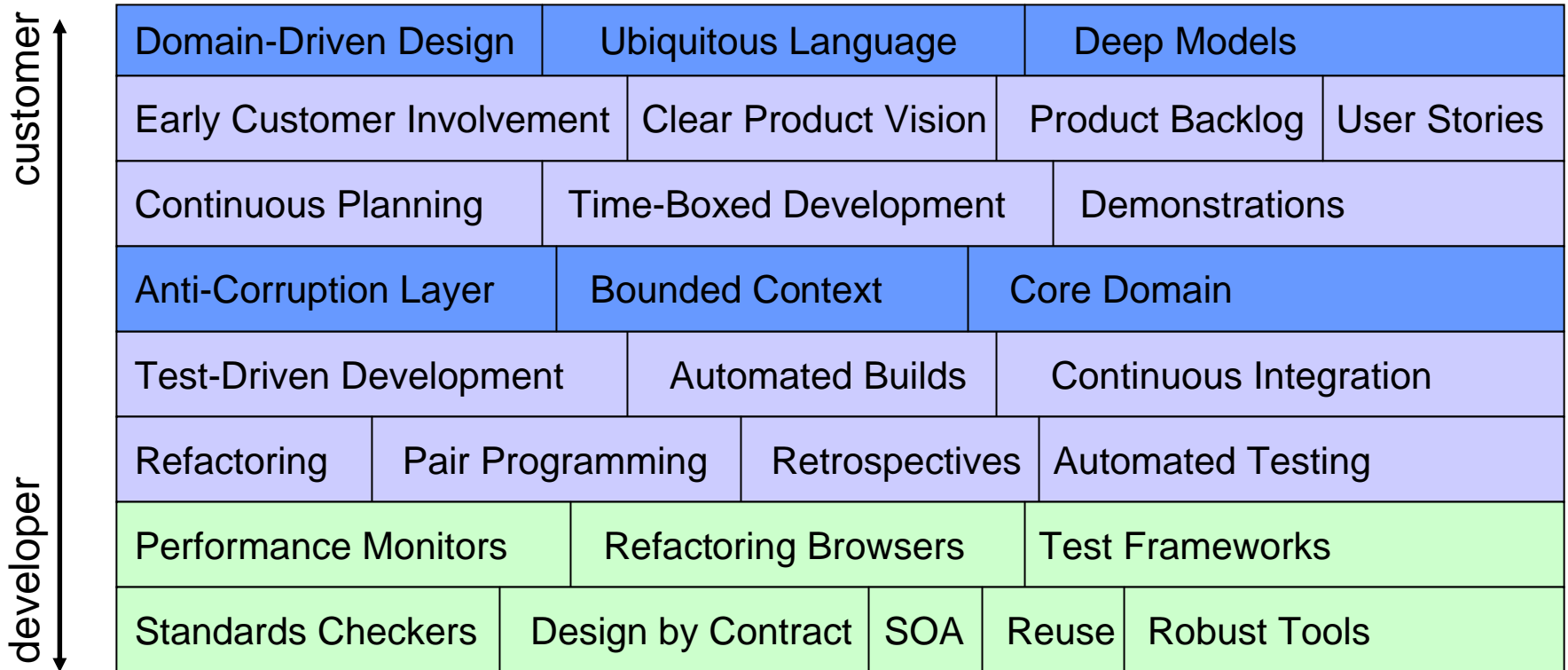## Some technologies help with the issue of solving the problem wrong.

5

The Better News

Some technologies help with the more difficult issue of solving the wrong problem.

6

The Best News

- **The technologies are synergistic**

| customer ↑		
Domain-Driven Design	Ubiquitous Language	Deep Models
Early Customer Involvement	Clear Product Vision	Product Backlog / User Stories
Continuous Planning	Time-Boxed Development	Demonstrations
Anti-Corruption Layer	Bounded Context	Core Domain
Test-Driven Development	Automated Builds	Continuous Integration
Refactoring / Pair Programming	Retrospectives	Automated Testing
Performance Monitors	Refactoring Browsers	Test Frameworks
Standards Checkers	Design by Contract / SOA / Reuse	Robust Tools
developer ↓		

- **The barriers to adoption are relatively low**
  - **Good FOSS tool support to get started**
    - **COTS tools also available with additional capabilities**
  - **Adequate literature and experience base for training**

Who is Involved?

- Who is involved in making a change to agile and domain-driven design?
  - Everyone!
    - Customers, Users, Systems Engineering, Software Development, Test, Specialty and Support Disciplines, Management, etc.
  - Software developers usually like the change because it is a more natural way to solve problems
  - Systems engineers and managers sometimes have a harder time adapting
    - Managers fear the loss of the perception of being in control
    - Systems engineers sometimes struggle with the need to keep the big picture in mind while going deeper into selected areas for near-term development
- Benefits generally outweigh the negatives
  - Earlier feedback on requirements, architecture and design
  - Earlier visibility into problem areas
  - Good vehicle for transferring domain knowledge

Some Experiences (1)

- Program applying many of the practices
  - Program Characteristics:
    - Mission critical technical application, critical algorithms
    - Adopted both domain modeling and agile practices
    - Part of a larger system doing traditional development
  - Results:
    - High quality product, with good quality measurements
    - Able to make substantial change to add capability and improve performance with very little impact
    - Happy engineering team and happy customers
    - Practices spreading to other teams
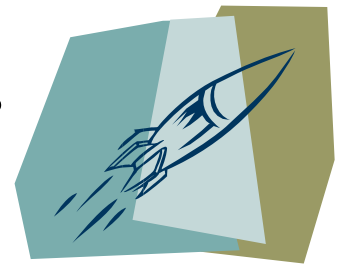
Some Experiences (2)

- Program applying just a few of the agile practices
  - Program characteristics
    - New capability being added to large existing system
    - Many new developers
    - Experts still involved in previous release that was behind schedule
    - Focused on standards, daily status meetings, continuous integration, refactoring, automated builds
  - Results
    - New capability developed on time and budget
    - High quality code, based on early test results and independent quality assessment
    - Happy team and happy customers

Summary

- Domain-driven design and agile development together offer substantial opportunities for improving how we do business
  - Tool support is now robust enough to support iterative development
  - Adequate material is available for training
  - Substantial and sustained improvements are becoming evident

Questions

Backup Material

- Descriptions of the Key Technologies
  - Description
  - Benefits
  - References
    - Additional descriptive material
    - A book for further study

Agile Development

- Description:
  - An approach to software development that uses short, time-boxed iterations to support early delivery of the customer's highest value capability
    - Each iteration is potentially shippable
  - Agile approaches use continuous planning, analysis and design rather than completing those activities up-front, before development begins
    - Customer is involved in prioritizing and clarifying requirements throughout each iteration
  - Examples:
    - Scrum, XP, Disciplined Agility, FDD, Adaptive Project Management
- Benefits:
  - Produces early value for customers
  - Accommodates changing requirements
  - Improves quality and productivity
- References:
  - http://en.wikipedia.org/wiki/Agile_software_development
  - Agile Software Development: The Cooperative Game by Alistair Cockburn

Test-Driven Development

- Description:
  - An approach to development that uses tests to drive the production of SW
  - Write a test, write the code, run the test, refactor
  - Examples:
    - Test frameworks include the xUnit family of FOSS tools (JUnit, cppUnit, nUnit, etc) as well as commercially available tools
- Benefits:
  - Produces high quality code with good interfaces and few dependencies, which improves the maintainability of the system
  - Makes future changes easier since all code has a suite of tests
- References:
  - http://en.wikipedia.org/wiki/Test-driven_development
  - Test-Driven Development: By Example by Kent Beck

15

Continuous Integration

- Description:
  - With continuous integration, developers check in their code several times a day, as soon as they complete each small chunk of functionality
  - When the code is checked in, a series of automated tests are run to ensure that both the new code and the existing code base function as expected
- Benefits:
  - Defects are discovered soon after they are introduced, so they are easier to find and fix
  - Fewer unpleasant surprises late in development
- References:
  - http://en.wikipedia.org/wiki/Continuous_integration
  - Continuous Integration: Improving Software Quality and Reducing Risk by Duvall, Matyas and Glover

Time-Boxed Development

- Description:
  - Each short iteration is scheduled for a specified duration – usually from 2-4 weeks
  - If the scheduled work cannot be completed, it is deferred to the next iteration
- Benefits:
  - Establishes a project rhythm that improves productivity
  - Provide early and frequent status based on working code
  - Forces hard choices about capability
    - The content must be allowed to change since schedule and quality are fixed
- References:
  - http://en.wikipedia.org/wiki/Timebox
  - Agile Project Management: Creating Innovative Products by Jim Highsmith

Automated Tests

- Description:
  - Test automation can occur at any level of testing
    - Unit test automation is supported through test frameworks like JUnit
    - Both FOSS and COTS products are available for automating aspects of higher level testing
  - Automated tests are designed to be run frequently, so they must be fast and free of side effects
    - Usually automated unit tests are run as an integral part of development (see TDD) and each time the code is checked into the CM system
- Benefits
  - Improved feedback to the developer and increases the quality of changes to the code
  - Automated tests are especially valuable for regression testing
  - Provide a safety net for future changes
- References:
  - http://en.wikipedia.org/wiki/Automated_testing
  - http://www.junit.org/
  - Fit for Developing Software: Framework for Integrated Tests by Rick Mugridge and Ward Cunningham

Early Customer Involvement

- Description:
  - Customers create and prioritize items in the product backlog
  - Daily interaction between customer and developers both clarifies requirements details and allows for the deeper understanding that helps manage the complexity associated with most domains
  - Examples:
    - Business person working with developers to clarify requirements for a payroll system or invoice system
    - Hardware engineer working with developers to clarify interactions between new hardware and controlling software
    - Systems engineers working with developers to develop algorithms for scheduling access to specific resources
- Benefits:
  - Reduces the need to capture requirements detail early in the life cycle
- References:
  - http://en.wikipedia.org/wiki/Extreme_Programming_Practices#Whole_team
  - Extreme Programming Explained: Embrace Change (Second Edition) by Kent Beck and Cynthia Andres

Demonstrate Each Iteration

- Description:
  - At the end of each 2-4 week iteration, the features developed during that iteration are demonstrated to the customer
    - Customer feedback drives priorities for future iterations
- Benefits:
  - Promotes better understanding of additional or different needs
  - Working code demonstrates real progress
- References:
  - http://www.scrumforteamsystem.com/ProcessGuidance/Process/SprintReview.html
  - Agile Software Development with Scrum by Ken Schwaber and Mike Beedle

User Stories

- Description:
  - Short descriptions of features that can be implemented in 2 days to 2 weeks
  - Three pieces to a user story
    - Short written description
    - Conversations about the story to flesh out the details
    - Tests that convey and document details and that can be used to determine when a story is complete
- Benefits:
  - Provide a good basis for estimating size as well as a mechanism for understanding user needs
  - Force a shift to verbal communication for feature details, which is much higher band-width and supports rapid feedback cycles
  - The tests associated with the stories provide executable documentation of user requirements
- References:
  - http://en.wikipedia.org/wiki/User_story
  - User Stories Applied for Agile Software Development by Mike Cohn

Product Backlog

- Description:
  - A product backlog is a prioritized list of features desired for a product
    - Grows and changes over time as more is learned
    - Scope line shows how much can be accomplished with current funding
  - Prioritization is primarily based on customer needs, but must also consider technical dependencies
  - Commercial and FOSS tools are available to help manage both the product backlog and iteration backlogs
- Benefits:
  - Prioritized development of functionality maximizes customer value
  - Allows users to add, subtract or change features based on new needs
  - Scope line provides on-going visibility into features that can be developed with current funding
- References:
  - http://www.mountaingoatsoftware.com/product_backlog
  - Scaling Software Agility: Best Practices for Large Enterprises by Dean Leffingwell

Frequent Deliveries

- Description:
  - Short iterations allow early delivery to customer
    - Each iteration should be potentially shippable
  - Often multiple iterations are grouped for delivery to customer
    - Functionality is complete with each iteration, but there may be need for a 'hardening' iteration before shipment to address publication of user documentation, final system tests, etc
- Benefits:
  - Allows customers to get early benefit from the system
  - Use of the system in the customer environment gives improved opportunities to discover 'the real requirements'
- References:
  - http://www.stsc.hill.af.mil/CrossTalk/2002/10/mccabe.html
  - Lean Software Development, An Agile Tool Kit by Mary and Tom Poppendieck

Clear Product Vision

- Description:
  - Product vision is established early to guide future development efforts
    - Essential when requirements are not fully detailed initially
  - One technique to establish the vision is to design the 'box' for the product
    - Differences between boxes designed by different teams can illuminate areas of disagreement on product priorities
- Benefits:
  - Helps focus customers and developers on the essentials of the product
  - Guides lower level implementation decisions
- References:
  - http://www.innovationgames.com/game/PRODUCTBOX.aspx
  - Agile Project Management: Creating Innovative Products by Jim Highsmith

Automated Builds

- Description:
  - Goal is to reduce the build process to a simple us-of-a-button action
    - Every programmer can perform a build whenever it is needed
      - Incremental builds can help make this a reality
    - With today's tools it is possible for even the largest systems to build multiple times a day
  - Enables effective Test-Driven Development
- Benefits:
  - Reduces time programmers spend on repetitive build tasks
  - Improves ability to run tests for every change, which improves quality and productivity
- References:
  - http://www.electric-cloud.com/solutions/agile_software_development.php
  - Integrating Agile Development in the Real World by Peter Schuh

Pair Programming

- Description:
  - Two individuals work side-by-side, sharing a single workstation, to design or code
  - Pairing with testers and engineers can be beneficial when requirements clarification is needed
- Benefits:
  - Provides a real-time peer review
  - Good mechanism for knowledge transfer
  - Improves code quality
- References:
  - http://en.wikipedia.org/wiki/Pair_programming
  - Pair programming Illuminated by Laurie Williams and Robert Kessler

Retrospectives

- Description:
  - At the end of each iteration, each team meets to celebrate the completion of the iteration and to capture lessons learned for the next iteration
  - Three topics to consider
    - What worked well and should be continued
    - What should the team stop doing
    - What needs to be done differently
- Benefits:
  - Identifies areas for improvement in the next iteration
  - Improved morale when team members know that the are listened to
- References:
  - http://www.retrospectives.com/
  - Agile Retrospectives, Making Good Teams Great by Esther Derby and Diana Lawson

Domain-Driven Design

- Description:
  - A way of accelerating software projects that have to deal with complex domains
  - Fundamental principles
    - The primary focus should be on the domain and domain logic
    - Complex domains should be based on a model
  - Agile approaches enable domain-driven design
    - Work with customers to develop models that reflect domain concepts
    - Provide rapid feedback to clarify complex areas
- Benefits:
  - Deeper understanding of the domain
  - Better communication between developers and domain experts
  - Ability to make breakthroughs at a faster pace
- References:
  - http://domaindrivendesign.org/
  - Domain-Driven Design: Tackling Complexity in the Heart of Software by Eric Evans

Ubiquitous Language

- Description:
    - A common language, based on the domain model, that serves as a communication vehicle between engineers, developers and domain specialists
        - Use of model-based terms in all project communication facilitates deeper understanding of the domain by everyone
        - One of the best ways to refine a model is to explore with speech, trying out loud various constructs from possible model variations
- Benefits:
    - Improved communication, which results in better models and better software
    - Helps in the discovery of hidden concepts
        - Often these arise in areas where the language does not flow smoothly
- References:
    - http://domaindrivendesign.org/discussion/messageboardarchive/UbiquitousLanguage.html

Deep Models

- Description:
    - An incisive expression of the primary concerns of the domain experts and their most relevant knowledge
        - A deep model sloughs off superficial aspects of the domain and naive interpretations
        - A deep model distills the most essential aspects of a domain into simple elements that can be combined to solve the important problems of the application
- Benefits:
    - Deep models enable acceleration of discovery and innovation within a domain
    - Keeps entire project on the same page
- References:
    - Domain-Driven Design: Tackling Complexity in the Heart of Software by Eric Evans

Core Domain

- Description:
  - The distinctive part of the model, central to the user's goals, that differentiates the application and makes it valuable
    - Efforts to refine and distill models should be focused on the core domain
- Benefits:
  - Identification of core, supporting and generic domains can help drive the company's strategy for what they develop, outsource or purchase
  - Helps identify the impact of changes
- References:
  - Domain-Driven Design: Tackling Complexity in the Heart of Software by Eric Evans

Bounded Context

- Description:
  - Defines the scope of each domain model
    - Identifies what has to be consistent and what can be developed independently
    - Defines the boundaries for continuous integration
  - Relationships between contexts can take multiple forms
    - Shared kernel, customer/supplier development teams or conformist
- Benefits:
  - Clearly identifies boundaries, which improves ability to integrate across teams
  - Understanding of relationships between contexts drives appropriate program behavior
- References:
  - Domain-Driven Design: Tackling Complexity in the Heart of Software by Eric Evans

32

Anti-Corruption Layer

- Description:
  - Allows new models to interface with legacy systems, without losing the clarity needed for deep modeling
  - Creates an isolation layer so that the new model can avoid corruption caused by needing to adapt to the semantics of the old system
  - Can be implemented by a combination of façade and adapter patterns, but it is more sophisticated than either of those
- Benefits:
  - Keeps one side of a bounded interface from leaking into the other, so the new models are not corrupted
    - Provides a translation between parts of the system that adhere to different models
- References:
  - http://domaindrivendesign.org/practitioner_reports/peng_sam_2007_06.pdf

Design by Contract

- Description:
  - An approach to software design that makes pre- and post-conditions explicit for each public method
    - Uses asserts (or equivalent) to enforce the contracts
  - Defensive programming is used at system boundaries, but not for interfaces within a boundary
- Benefits:
  - Interface mismatches are detected immediately, rather than indirectly through the errors that result from the mismatch
- References:
  - http://en.wikipedia.org/wiki/Design_by_Contract
  - Object-Oriented Software Construction, Second Edition, by Bertrand Meyer

Patterns

- Description:
  - A pattern is a repeatable solution to a common problem in software design
    - Captures lessons learned about use of the solution in various situations
  - Catalogs of patterns exist at various levels, including architecture patterns and design patterns
- Benefits:
  - Leads to higher quality designs that are easier to maintain with fewer dependencies
  - Enhanced communication of intent, based on the pattern selected
    - The pattern name conveys a lot of information in a few words
- References:
  - http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29
  - Design Patterns: Elements of Reusable Object Oriented Software by Gamma, Helm, Johnson and Vlissides

Refactoring

- Description:
  - An approach that systematically changes the internal structure of code without changing its behavior
  - An integral part of test-driven development
  - Often done prior to making major changes to code, in order to make it easier to make the changes
    - Each small change is tested so any errors are detected immediately
- Benefits:
  - Cleaner code, fewer dependencies, fewer defects
  - Supported by refactoring browsers, so it is a relatively safe way to make code changes
- References:
  - http://en.wikipedia.org/wiki/Refactoring
  - Refactoring: Improving the Design of Existing Code by Martin Fowler

SOA and Reuse

- Description:
  - SOA is an architectural style where existing or new functionalities are grouped into atomic services
    - The goal of SOA is to allow fairly large chunks of functionality to be strung together to form ad-hoc applications which are built almost entirely from existing software services
- Benefits:
  - Supports large-grained reuse of independently developed services
    - Enhanced productivity and quality through reuse
  - Enables increased focus on the core domain
- References:
  - http://en.wikipedia.org/wiki/Service_oriented_architecture
  - Service-Oriented Architecture: Concepts, Technology and Design by Thomas Erl

Robust Tools

- Description:
  - There are many tools available today that actually help in the development of software
    - Standards checkers, and standards checking services, Refactoring browsers, Test frameworks, Performance monitors, Modeling tools, especially those with reverse engineering capabilities
  - Need to make sure that the selected tools do not drive extra work
- Benefits:
  - Improved developer productivity
  - Improved quality
  - Enablers for iterative development – manual processes not adequate to support short development cycles
- References:
  - http://www.opensourcetesting.org/functional.php
  - http://www.opensourcetesting.org/unit_java.php
  - http://www.ddj.com/TechSearch/searchResults.jhtml;jsessionid=MPJGHOHFWKVKCQSNDLOSKH0CJUNN2JVN?queryText=tools