

# Test Driven Design Methodology for component based systems

Baldev S. Mattu: [bsmhsm@hotmail.com](mailto:bsmhsm@hotmail.com)

Proff. Ravi Shanker: [ravi@cse.fau.edu](mailto:ravi@cse.fau.edu)

Center for Systems Integration  
Department of Computer Science and Engineering  
Florida Atlantic University

April 10, 2007

# Outline

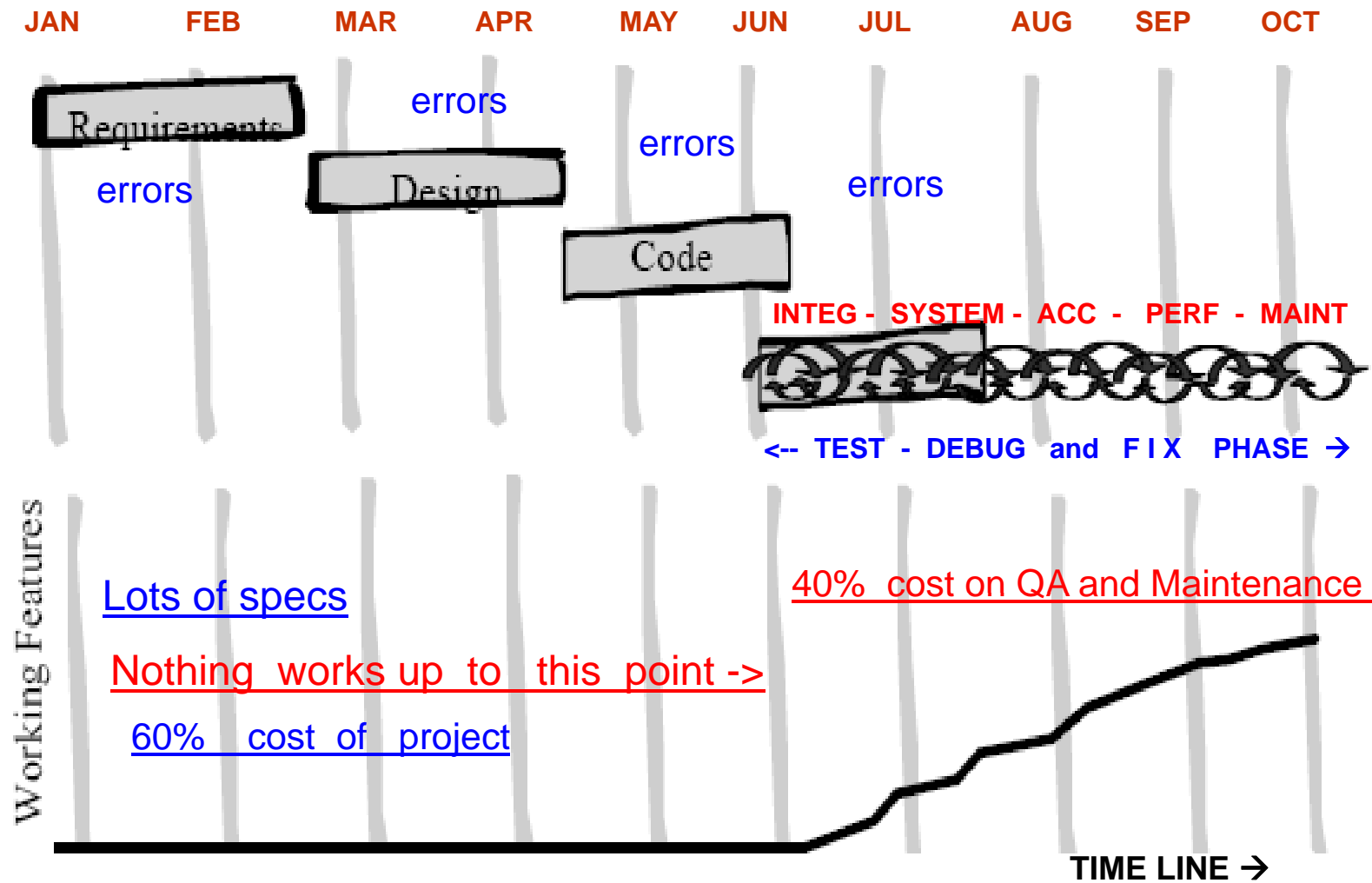
- Traditional Development Method
- TDD – Test Driven Design Method
- Best practices
- TDD Cost of change Curve
- Evidence from Industry Investigations
- TDD Limitations and Recommendations
- Future Work
- Benefits/Summary
- References

# Current Design Methodology

- **Traditional Development is phased, One phase follows another**
  - Big Gap between problem area and the solution area
  - Returning to a phase after it is complete is not allowed
  - Cannot add new customer requirements late in the development process
- **Has flaws**
  - It's nearly impossible to anticipate all of the system requirements before design and implementation
  - Software not isolated, Dependencies, Dead & Duplicate & Complex Code
  - The riskiest phase, testing, occurs late in the process
  - **BUGS and HIGH COSTS of maintenance**

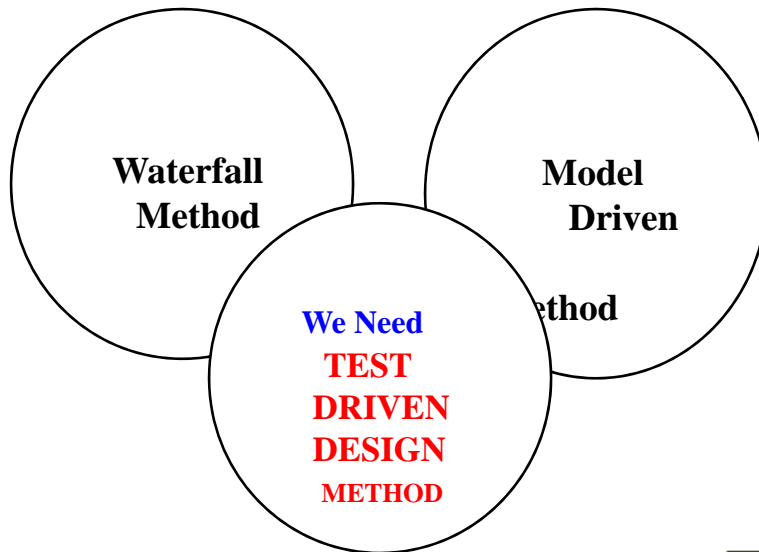
# Current Software Design Method

- Outdated Software Development Practice



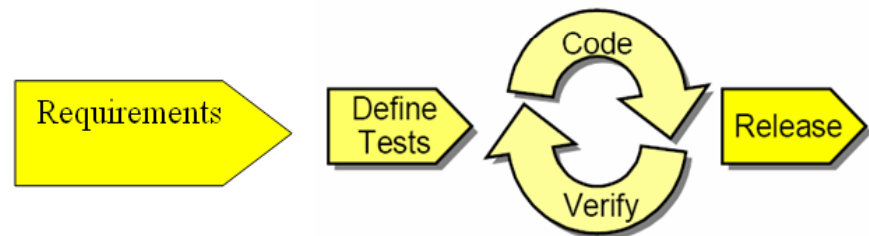
# Do we need something else

## Water Flow Method has flows



## MDD Issues of Use Cases

- The customer doesn't understand the use cases
  - The system boundary is undefined
  - There are too many use cases
  - The use-case specifications are too long
  - The use-case specifications are confusing
  - The use cases are never finished
- by Susan Lilly, 2000



**Test Driven Design/Test First Development**

# TDD – Test Driven Design Method

## - what is it

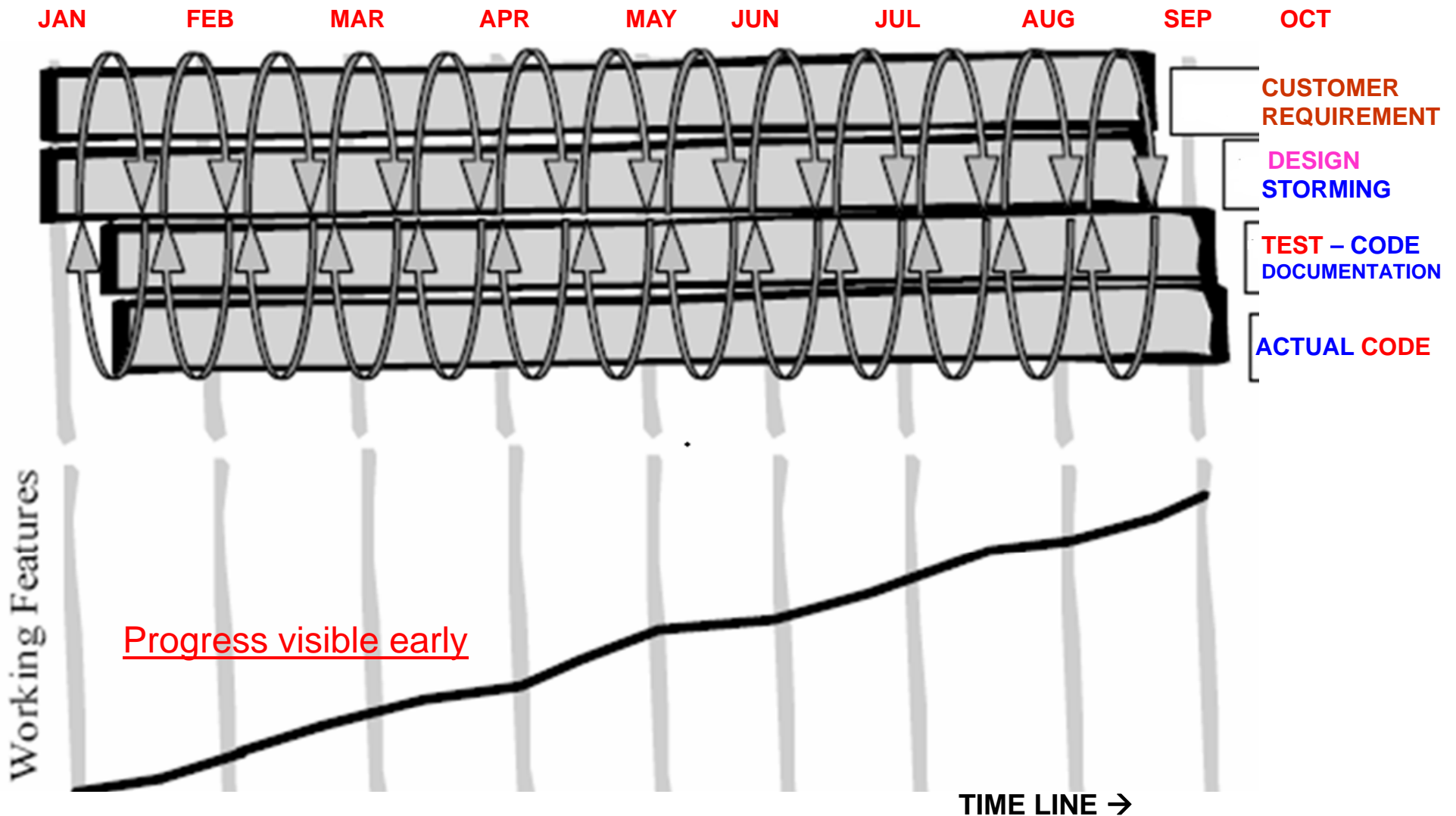
- **Evolutionary, Iterative and Incremental Approach to development**
  - From system requirement specification to many customer requirements/user stories
  - De-emphasize big up front designs
  - Developer is lead to think: about how to use the design, why do we need it, what it is for
- **Design is constantly addressed in sequence by**
  - Brain storming and specifying design details
  - Writing test code
  - Writing production code
  - Improving existing code design via refactoring
- **Goal is to quickly move to development in baby steps**

# TDD - Players/Roles

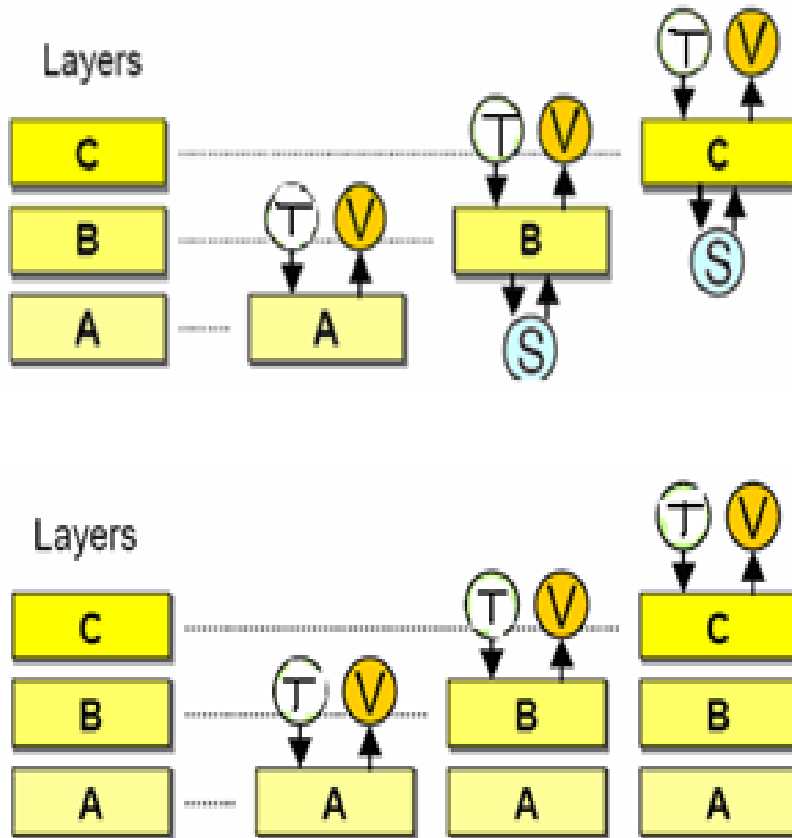
- **Customer / Business / User / Domain Expert**
  - Specifies the requirements for system coverage
  - Focusing on the scenarios, the flow of events, the dynamic behavior
  - Validates the requirements have been met before system release
  - Must define conventional tests for each iteration
- **Development start when features are**
  - Defined to a reasonable level of granularity
  - Designed to some fixed level of confidence
  - Prioritized according to dependencies and importance mapped out for finite requirements.
- **TDD Developers – up to twelve programmers**
  - Code estimate, Unit testing, Acceptance testing, Integration or Component testing, Write test & code for bugs found

# Test Driven Design Iteration

Evolutionary – Iterative - Incremental Approach



# TDD - Unit Testing & Continuous Integration of Components



- **Unit Testing - Bottom-up**
- eliminate bugs early before units are combined into components and components into systems
- To prove that your code actually works
  - Immediate Confirmation that things are working
  - It's even better than code inspection
- XUnit family of tools
  - JUnit for unit testing
  - DBUnit for component testing
  - JUnitPerf for performance tests
  - ECLIPSE for Refactoring
  - ECLIPSE with JUnit built in

- **Integrate Components - Bottom-up**
- Start integration early to reduce risk
- Use all automated unit tests for continuous integration via
  - ANTS, Cruise Control
  - ECLIPSE with ANTS and Cruise Control

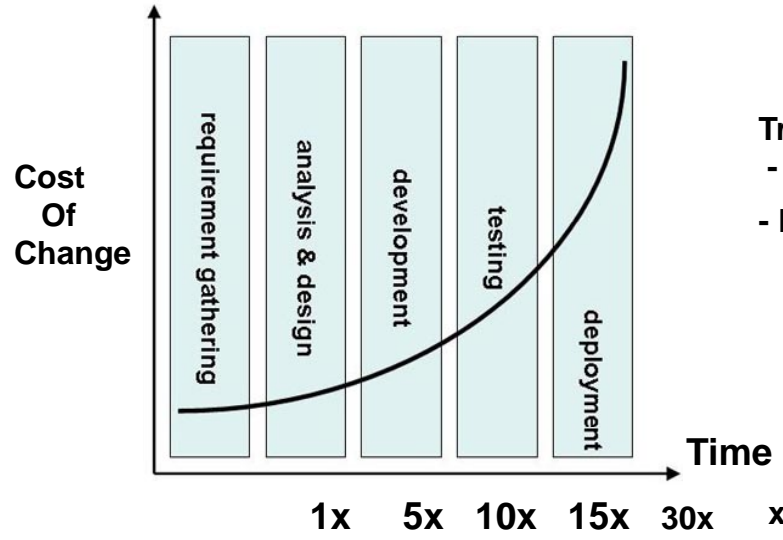
# TDD – Acceptance Testing

- Customer/User writes one or more acceptance test for each requirement
- Acceptance tests are executed in each iteration
  - To understand progress of the system behavior
  - To verify that the system is ready for release
- Targets errors not found in unit testing
  - Requirements are mis-interpreted by developer
  - Modules don't integrate with each other
- Tools for Acceptance testing of functions
  - FitNesse - RTF metric
  - FitNesse ECLIPSE plugin

# TDD - Best practices

- Use pair programming
- Generate code skeletons
- Decouple your tests
- Facilitate early and continual integration
- Recent and Mature Tests
- Everyone might not be taking a TDD approach

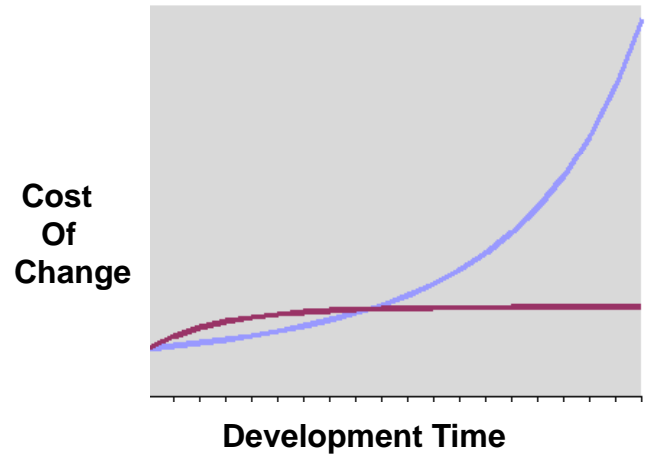
# Cost of Change in Traditional and TDD



**Traditional Cost of Change**  
- Methods & Tools issue, Spring 2005  
- Later changes are more expensive

x is a normalized unit cost in terms  
Of person-hours, dollars, etc.:  
Gregory Tassej 2002 , Gatherburg

**Cost of Change vs Development Time**

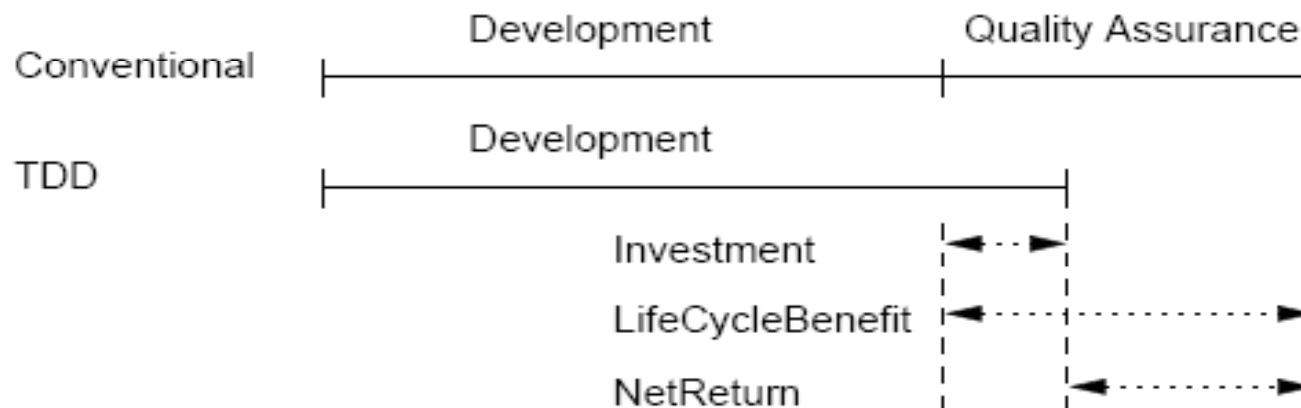


**Traditional Cost of Change**

**Kent Beck's cost of change curve**  
- Each change cost similar amount

# Evidence – TDD Investigations in Industry

- **Initial Investigation of TDD in Industry by Boby George – 2003**
  - TDD facilitates simpler design and lack of upfront design is not a problem
    - Increases programmer's productivity
    - Process takes longer but code quality goes up
- **IBM Retail Store Solutions study by E. Michael Maximilian**
  - found defect rate reduced by 50% after Functional Verification Test when using TDD method
- **TDD's Return on Investment – Benefit Overview by Muller & Padberg**
  - $ROI = ( LifeCycleBenefit - Investment ) / Investment$ 
    - If investment pays off, ROI is positive, otherwise negative
    - Benefit of TDD = to the QA phase of the conventional method



# TDD - Unit Testing Limitations

- Does not catch Integration errors, performance problems or system wide issues
- TDD does not cover Interactions between functional units
- TDD has to be learned
  - Continual Training for Developers and Testers
  - Unit testing not taught enough to develop good habits
- TDD can test untested legacy code, Russ Harold-2006
  - May introduce new bugs

# TDD – Recommendations

- Management must buy into the new approach
- Requires change in doing development
- Combine TDD with conventional test methods
  - TDD alone is not enough
- Use pair programming for best results
- Refactor often to optimize code
- Combine MDD and TDD

# TDD – Future Work

- Explore possibility of TDD with other technologies
  - Model Compiler
  - CORBA Component Model
  - Rhapsody
  - MSC – message sequence charts
  - LTSA for concurrency issues
- Develop RhapsUnit Test framework TDD for Rhapsody environment

# TDD - Benefits / Summary

- System-level documentation of Design
- What TDD excels at is getting the business functionality and mainline logic correct
- Late changes can be made quickly
- Users get more features rapidly
- High test coverage with fast feedback
- TDD leads to increase in programmer's productivity and higher code quality

# References

- Preliminary Analysis of the Effects of pair Programming and Test-Driven Development on External Code Quality, Lech Madeyski
- Test Driven Development – A Practical Example, *Dave Astels*
- Continuous Integration – last update: 01 May 06  
<http://www.martinfowler.com/articles/continuousIntegration.html> *Martin Fowler*
- Assessing TDD at IBM: by E. Michael Maximilian  
<http://www.ipd.uka.de/Tichy/uploads/folien/122/maximilienICSE03.pdf>
- An Initial Investigation of TDD in Industry: by Boby George  
<http://collaboration.csc.ncsu.edu/laurie/Papers/TDDpaperv8.pdf>
- About the Return on Investment of TDD: by Matthias Muller  
<http://www.ipd.uka.de/mitarbeiter/muellerm/publications/edser03.pdf>
- <http://ant.apache.org/>
- <http://cruisecontrol.sourceforge.net/>
- <http://www.junit.org/index.htm>
- <http://fitnesse.org/>
- <http://www.eclipse.org/>