Agile Software Development

Venkat Subramaniam
venkats@agiledeveloper.com

July 2004

Presentation and examples can be downloaded from
http://www.agiledeveloper.com/download.aspx

Abstract

Abstract Agile Software Development approaches emphasize test first coding, refactoring, paired programming. Brute-force coding ignoring design principles may lead to system that is hard to extend and maintain. However, systems built using principles at upfront design may lead to needless complexity as well. This talk will use practical examples to illustrate good development practices and tools. Emphasis will be placed on using test first coding, continuous integration, various principles to be learnt and followed all though the development cycle. The attendees will participate in designing and developing an application during the session. The code developed will be made available for free download on the speaker's web site.

Speaker Dr. Venkat Subramaniam, founder of Agile Developer, Inc., has taught and mentored more than 2,500 software developers around the world. He has significant experience in architecture, design, and development of distributed object systems. Venkat is an adjunct professor at the University of Houston and teaches the Professional Software Developer Series at Rice University's Technology Education Center. He may be reached at venkats@agiledeveloper.com.

Examples Any page with a  has an example attached
Download from http://www.agiledeveloper.com/download.aspx

Agile Software Development

- **Let's Design a System**
- Implementation of the System
- Agile Development Practices
- Planning
- Test First Coding
- Writing Tests
- Refactoring
- Continuous Integration
- Conclusion

Tick-Tack-Toe

A small assignment for you. For the problem given below, come up with ideas of how you may implement it. Draw UML class diagram(s) and write a paragraph (or two) explaining how you would implement it. No need to actually code it at this time. Then we will embark on implementing this during the session.

There are two users to the system. One will place an 'x' peg and the other an 'o' peg in cells. There are three rows and three columns. First a user must indicate whether first player will use the 'x' peg or the 'o' peg. Then the first player is asked to place a peg on a cell. The player can only place on an empty cell. The game continues until a player has placed three pegs in a row, column or diagonally or there are no more empty cells left. If the game is won, the victory is announced. The application will keep track of the number of wins by each player. At any time, a user may request to view the statistics of the name of players and number of games each one has won.

Intentionally left blank (use to sketch your ideas)

Agile Software Development

- Let's Design a System
- **Implementation of the System**
- Agile Development Practices
- Planning
- Test First Coding
- Writing Tests
- Refactoring
- Continuous Integration
- Conclusion

Implementation

- Paired Programming to implement the Tick-Tack-Toe application
- Available for download within 24 hours of presentation at http://www.agiledeveloper.com/download.aspx

Quiz Time

Agile Software Development

- Let's Design a System
- Implementation of the System
- **Agile Development Practices**
- Planning
- Test First Coding
- Writing Tests
- Refactoring
- Continuous Integration
- Conclusion

Development Goals

- To minimize the risk in development
  - Understand requirements better
  - Be ready to change as requirements change
- To succeed in the development process
- To complete the project
  - in budget
  - on time
- If the project has to be cancelled, do so with minimal damage
- Create a system that is
  - easier to maintain
  - less expensive to evolve
- Keep the bug count low

What about extensibility?

- Your system should be able to change with least cost
- You should anticipate change?
- Does it mean that you build for what you think may be needed?
- It depends
- Here are questions to ask

Cost of the new feature

- What are the chances you will need to add new feature?
- How much does it take now to provide it?
- What is the worth of that feature to customer?
- How much will it cost to provide it in the future?
- If it will cost almost the same in the future, and you are not certain of the feature's worth, it may be better to wait
  - If the features are important, we can implement it later
  - If it is not needed, we did not implement it

So Should I not worry about extensibility?

- You should!
- However, there are ways to address it
- Check on your ability to anticipate the need and change
- Check on your ability to build the system so the change in the future is incremental
- Refactor the system as it evolves

Control Variables

- Cost
  - Too little, does not solve problems
  - Too much, some times more of a problem
- Time
  - More time can improve quality and increase scope
  - Too much time hurts as well
    - Feedback from system during development is imperative
- Quality
  - Sacrificing this may result in short term gains
  - Over the long haul, lost is enormous
- Scope
  - Lesser the scope, better the quality
  - You can deliver sooner as well
  - Assuming it meets the business needs

Set of Values

- Communication
  - Need to communicate critical change in req., design, etc.
  - Put in place practices that will enhance communication
- Simplicity
  - Find simplest thing that will work
  - Build some thing simple today and pay a little to change tomorrow than build some thing complicated today that may never be used
- Feedback
  - Unit tests provide feedback
  - Corrected in minutes and days, not weeks
  - A system that stays out of the hands of users is trouble waiting to happen
- Courage
  - Do not hesitate to throw code away if you find a better simpler way
  - Do not hesitate to call attention to problems if they are significant and will benefit from reworking

Agile Software Development

- Let's Design a System
- Implementation of the System
- Agile Development Practices
- **Planning**
- Test First Coding
- Writing Tests
- Refactoring
- Continuous Integration
- Conclusion

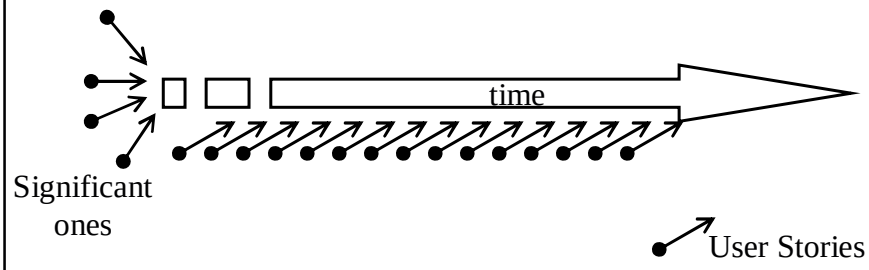
"Plans are nothing. Planning is everything,"
Dwight D. Eisenhower

"No plan survives contact with the enemy,"
Helmuth von Moltke

Planning

- It is more important to be successful in a project than staying with a plan
- Agile Software Practices focus on changing to suite the needs than sticking with a plan that has been developed

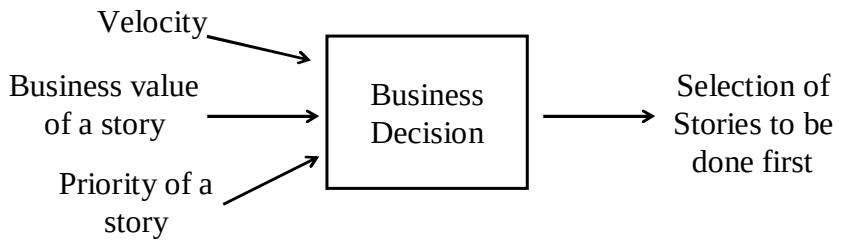
Development Process



Estimation

- Accurate estimation is hard
- Estimation comes from
  - Experience
  - Understanding the problem
  - Comfort with technology
  - Productivity
- Too big a story – harder it is to estimate
- May need to split it into more manageable pieces
- Velocity is the rate at which stories are implemented
- Spiking – Development of prototypes to get a feel for the velocity of the team

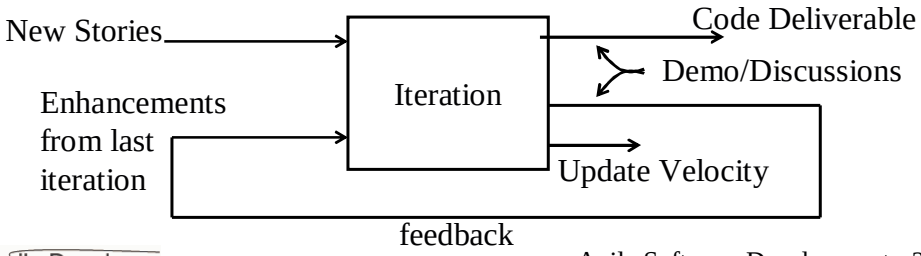
Release Planning



- Can't choose more stories than allowed by velocity
  - Based on velocity that is not accurate in the beginning
- As velocity is varied, this will vary as well

Iteration Planning

- Typically two weeks long
  - Personally I follow one week iteration
- Customer (and team) choose stories to be implemented for that iteration
  - based on velocity



Iteration Planning…

- Build Product and demo
- Do not build *"for"* demo
- Iteration ends on specified date
  - Even if some stories are not done

Agile Software Development

- Let's Design a System
- Implementation of the System
- Agile Development Practices
- Planning
- **Test First Coding**
- Writing Tests
- Refactoring
- Continuous Integration
- Conclusion

How we typically create classes?

- We think about what a class must do
- We focus on its implementation
- We write fields
- We write methods
- We may write a few test cases to see if it works
- We hand it off to users of our code
- We then wait for them to come back with feedback (problems)

Test First Coding

- How about starting with a test case even before we have any code for our class?
- How about first write test that fail because the code to support it does not exist?
- How about adding functionality to our system by adding tests incrementally and then adding code to make those tests succeed?

Test First Coding Benefits

- It would
  - completely revert the way we develop
  - We think about how our class will be used first
    - Helps us develop better interfaces that are easier to call and use
  - Would change the way we perceive things
  - Will have code that verifies operations
  - Will increase robustness of code
  - Will verify changes we make
  - Will give us more confidence in our code

Test First Coding Benefits…

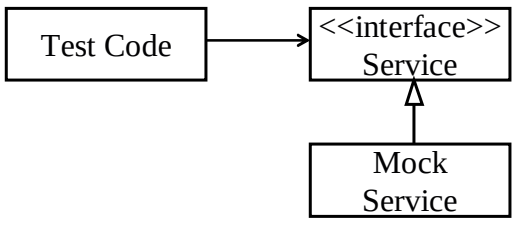
- Forces us to make our code testable
- Tests decouple the program from its surroundings
- Serves as invaluable form of documentation
  - Shows others how to use our code

Test Isolation – Mock Objects

- How do we create a test when our system may depend on
  - A database to persist information
  - A third party simulator to perform calculations/functions
  - A printer to print output
  - A scanner or device to read input?
- We may implement out system with Mock Objects

Mock Objects

- A Mock Object
  - Provides the expected functionality
  - Isolates the code from details that may be filled in later
  - Speeds up development of test code
  - Can be refined incrementally by replacing with actual code



Unit Testing

- Unit testing
  - Is more of an act of design than verification
  - Is more of an act of documentation than verification
  - Provides excellent feedback

Agile Software Development

- Let's Design a System
- Implementation of the System
- Agile Development Practices
- Planning
- Test First Coding
- **Writing Tests**
- Refactoring
- Continuous Integration
- Conclusion

Red/Green/Refactor

- First write a test code that fails
- Implement enough code to make the test succeed
  - Go ahead lie your way though it
  - Only so much you can lie
  - Keep it simple; do not complicate things
- Refactor the code to improve it

Stay one step from Green

- At any time, we should be one step away from a green bar
- Why?
  - Gives confidence with change
  - Let's us focus on one thing well
  - Avoids tendency to write up a bunch of test cases at one time

Where should your test go?

- Not only public methods are tested
- How do you test protected or package friendly methods?
- Test code should be in the same package as your class being tested

Isolate your Tests

- One test should not affect another test
- One test should not fail because another test failed
- Provides order independence
- You can pick arbitrary set of tests to run

Test First & Assert First

- When should you write a test?
- Before writing the code to be tested!
- Remember red/green/refactor
- Write your tests with Asserting for result/conditions in mind

Writing Test Stubs?

- What if you plan to implement an interface method later?
- You plan to leave a dummy implementation in place
  - You have no time for it now
- Why not write a Test that will fail as soon as the method is implemented?
  - Have the method throw an exception when called
  - In the Test, Assert for the receipt of that Exception

How good are your Test?

- Your test are not good if they
  - Have long setup code
  - Have setup duplication
  - Take long duration to run
  - Are fragile

Where to run?

- It is not enough to run your tests on developer's machine
- Tests should run on each platform supported by the product!
- Why?
  - You do not want to miss variations or differences in behavior on different platforms
  - Learning tests come in to picture here as well
- Consider continuous integration (discussed later)

OK, all said, we found a bug?!

- We have solid tests (or so we thought)
- A bug is found
- What should we do?
- Understand the bug and fix it, right?
- Nope
- First write a (missing) test case that will bring the bug to surface
- Get the red bar on it first
- Then fix the bug to get to the green bar
- Refactor as necessary

Tools for Unit Testing

- JUnit
  http://www.junit.org
- For testing Java UI
  - Jemmy
    http://jemmy.netbeans.org
  - JFCUnit
    http://jfcunit.sourceforge.net
  - Abbot
    http://abbot.sourceforge.net
  - Pounder
    http://pounder.sourceforge.net

Agile Software Development

- Let's Design a System
- Implementation of the System
- Agile Development Practices
- Planning
- Test First Coding
- Writing Tests
- **Refactoring**
- Continuous Integration
- Conclusion

What is Refactoring?

- The Process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure

- Why fix what's not broken?
  - A software module
    - Should function its expected functionality
      - It exists for this
    - It must be affordable to change
      - It will have to change over time, so it better be cost effective
    - Must be easier to understand
      - Developers unfamiliar with it must be able to read and understand it

What is needed for Refactoring?

- "Before you start refactoring, check that you have a solid suite of tests. These tests must be self-checking"

What to Refactor?

- Duplication of Code
  - Identical code should be unified
- Step-wise refinement
  - If changing format or protocol, make the change gradual and test along the way
- Extract Method
  - Break a long method by separating into a method and calling it
- Inline Method
  - To eliminate twisted logic or convoluted calls
- Extract Interface
  - You are generalizing and creating other implementations
- Move method
  Move the method to where it is more appropriate, may be more efficient
- Method Object
  - Make an object out of a method that requires several parameters and local variables

Agile Software Development

- Let's Design a System
- Implementation of the System
- Agile Development Practices
- Planning
- Test First Coding
- Writing Tests
- Refactoring
- **Continuous Integration**
- Conclusion

Continuous Integration

- What good are the test cases if they are not run
- How often should we run them?
- Every night at least
- How about once every hour?
- Or better still when ever code change is checked in
- When code is checked in the code is compiled automatically and all tests cases are executed
  - If a test fails the team is alerted
  - When test fails, nothing else important/high priority
    - Fix the code to make the test succeed
    - Or modify the test to fit the changes if appropriate

Tools for Continuous Integration

- Anthill
  http://www.urbancode.com/projects/anthill/default.jsp
- Cruise Control/Cruise Control.NET
  http://cruisecontrol.sourceforge.net/
- Draco.NET
  http://draconet.sourceforge.net/
- Gump
  http://jakarta.apache.org/gump/

Quiz Time

Agile Software Development

- Let's Design a System
- Implementation of the System
- Agile Development Practices
- Planning
- Test First Coding
- Writing Tests
- Refactoring
- Continuous Integration
- **Conclusion**

Conclusion

- We all want to write software successfully
- Only constant is change
- How to keep up with it?
- Paired programming & Collective Ownership is code review on steroids
- Unit Testing gives constant feedback
  - Test cases are my angels
- Continuous integration is imperative
- Refactoring and Testing is a design process
- Let's succeed in development

References

1. Agile Software Development, Principles, Patterns, and Practices, Robert Martin

2. Refactoring Improving The Design Of Existing Code, Martin Fowler

3. Test-Driven Development by Example, Kent Beck

4. Continuous Integration, Martin Fowler http://www.martinfowler.com/articles/continuousIntegration.html

5. Examples, slides are for your download at

http://www.agiledeveloper.com/download.aspx