Agile Software Development

Venkat Subramaniam

venkats@agiledeveloper.com

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

Abstract

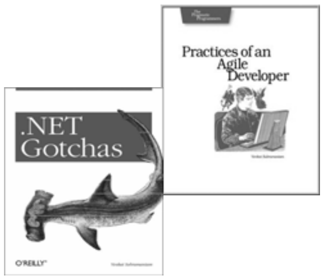
Abstract What is Agile software development? How should you change the way you develop your software? How do you plan? What about iterative development? What are some of the better practices that give results? In this session, the speaker will present various approaches that lead to a successful development. Tools that aid towards this goal will be highlighted as well.

We will discuss about project and iteration planning, test driven development, continuous integration and other practices that will help you realize agility on your projects.

About the Speaker *Dr. Venkat Subramaniam*, founder of Agile Developer, Inc., has trained and mentored thousands of software developers in the US, Canada and Europe. He has significant experience in architecture, design, and development of software applications. Venkat helps his clients effectively apply and succeed with agile practices on their software projects, and speaks frequently at conferences.

He is also an adjunct faculty at the University of Houston (where he received the 2004 CS department teaching excellence award) and teaches the professional software developer series at Rice University School of continuing studies.

Venkat has been a frequent speaker at No Fluff Just Stuff Software Symposium since Summer 2002.



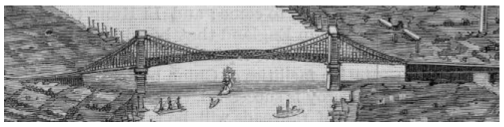
Agile Software Development

- **State of Software Development**
- Agility
- Planning
- Daily Activity
- Conclusion

Evolution of Fields

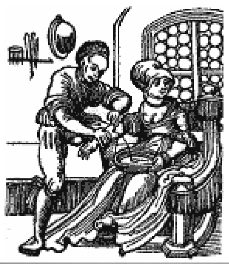
- Bridge Construction
- Medicine
- Airplanes
- Software Development

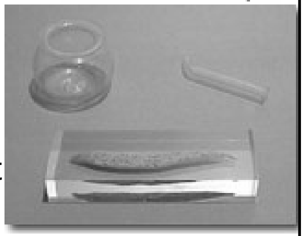
Bridge Construction



- Early Wood, Stone
- Then Iron, Steel
- Concrete Bridges
- Constructing a bridge is different from innovating a bridge (with new material for instance) for the first time
- Engineers use well established metrics to design bridges – they do not innovate at this stage

Medicine





- "Health was thought to be restored by purging, starving, vomiting or bloodletting"
  - Both surgeons and barbers were specializing in this bloody practice
  - Widely practiced in 18th and 19th century
  - Declared quackery by 1900
- Infection control
  - If patent survived surgery, he most likely died out of infection
  - Germ theory and sterility came only in late 1800s (Lister)
  - Current rate of infection < 2.5%

Airplanes

- 400 BC Chinese fly kite aspiring humans to fly
- For centuries, we tried to fly like birds… disastrous
- Steam powered, hot air
- Gliders, single man
- Engine powered
- 1903 Wright brothers' first flight – 12 seconds, 120 feet, 10 feet altitude

Software Development

- Relatively nascent field in comparison
- Machines are getting faster or more powerful
- Are we getting better in delivering software applications though

Success (or lack there of)

- How successful are we in developing software?

- Less than 10% of software projects succeed[1]
- Criteria for success?: On time, within budget, feature complete, works (failure free)

- Why is it so hard to get this right?

What's Software Development?

- Is it
  - Mathematics?
  - Logic?
  - Engineering?
  - Art?

- Combination of all of that[2]

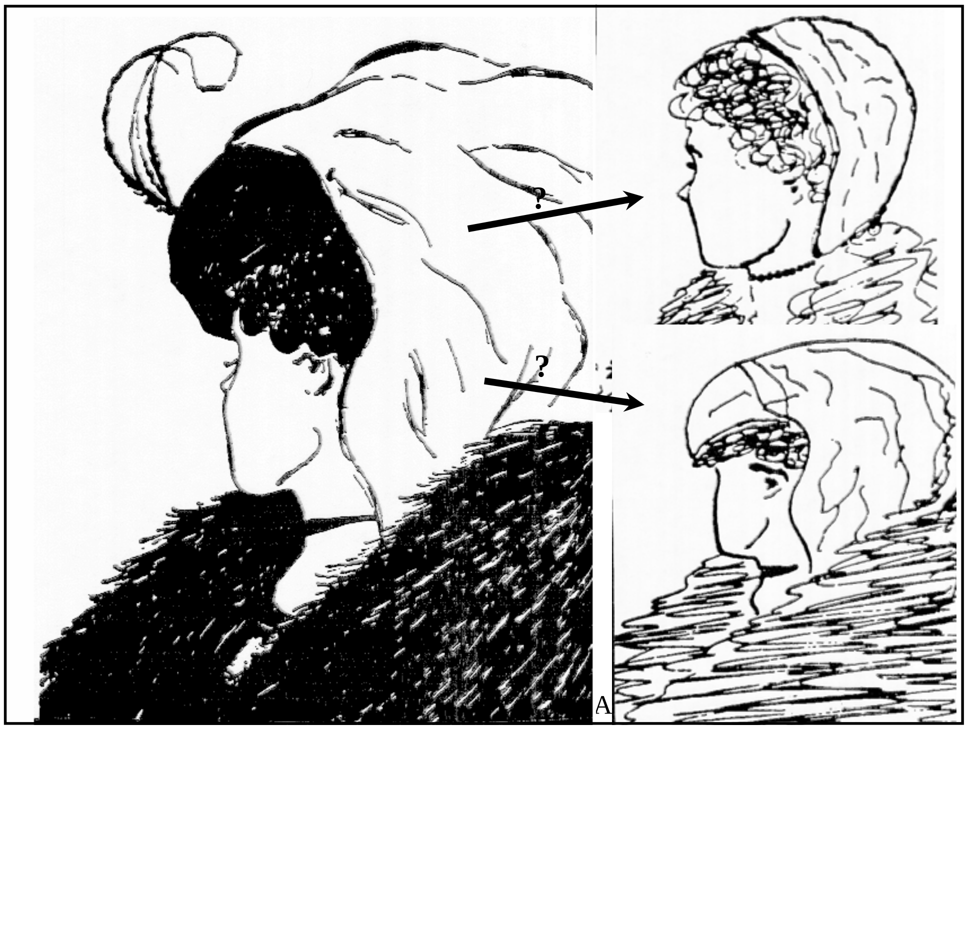
Software Engineering?

- What's Engineering?[2, 3]
  - "the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people"
  - "the design and manufacture of complex products <software *engineering*>"
- If software engineering like manufacturing or designing a manufacturing plant?
  - Is it like making another cell phone or making of cell phones (took 37 years for commercialization)?
- Manufacturing is predictive
  - You can measure and control quality, quantity
- Designing a manufacturing plant is creative/innovative
- Most software development is innovative process rather than predictive manufacturing
  - Requires great deal of innovation, interaction/communication

Why is it hard to communicate?

- Why not simply write good documents to describe requirements and hand them off to developers to create software?
- We have tried that, but we know it does not work
- 3 factors influence
  - What you are communicating
  - Who is communicating
  - With whom

- A Picture is worth a thousand words
- From Stephen Covey's "7 Habits of Highly Effective People"



Realizing what makes it hard

- Documents can’t fully describe the requirements
- 3 types of people make up your team
  - Those with exceptional domain knowledge but little software development expertise
  - Those with exceptional software dev. experience, but little domain knowledge
  - Those with both domain and software development skills
  - (we will ignore that 4th category)
- Closer and frequent interaction is a necessity

What are our 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

Agile Software Development

- State of Software Development
- **Agility**
- Planning
- Daily Activity
- Conclusion

Agility

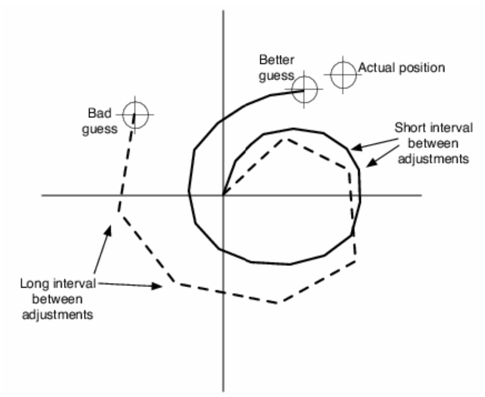
- What’s Agility?
- Being agile
- What’s Agile?
- “marked by ready ability to move with quick easy grace”
- “having a quick resourceful and adaptable character”
- What does that mean?
  - Process has to be lightweight and sufficient
  - Lightweight helps us adapt and move
  - Sufficient recognizes our ineffectiveness to be complete and relies on strong communication

Process

- Waterfall approach[4]
  - Actually specified iteration - largely ignored
- Customers' mind is not frozen after they give us the requirements
- We are not able to fully understand what is said
- Show me a long project duration, I will show you a project that is already doomed

Iterative and Incremental

- How to foster innovation and communication?
- Isolation does not help
- Interaction is key
  - among developers and with customers
- But will that not take more time?


Better guess
Actual position
Bad guess
Short interval between adjustments
Long interval between adjustments

The time/scheduling hypocrisy

- What can you tell me about the next project, you ask?
  - It is due on November 1st tells your manager

- We hold deadlines too dearly
- Of course, time to market is critical

- But what generally happens on projects when you hit that deadline?

Pick Two

- Ask your customers to pick two out of the following, you decide the third:

- Time
- Scope
- Quality

- Reality often ignored in project planning

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?[5]
- 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

Being Agile without going overboard

- How to move towards agility?

- Some practices that have strong impact
  - Planning
    - Shorter iteration cycle/ planning
  - Daily Activity
    - standup meetings
    - Unit Testing
    - Refactoring
    - Continuous integration

Agile Software Development

- State of Software Development
- Agility
- **Planning**
- Daily Activity
- Conclusion

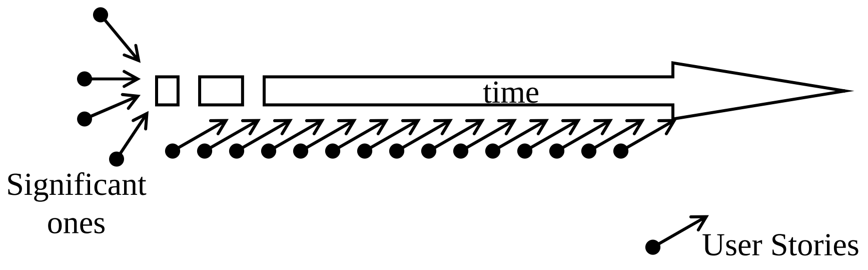
"Plans are nothing. Planning is everything," Dwight D. Eisenhower

“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

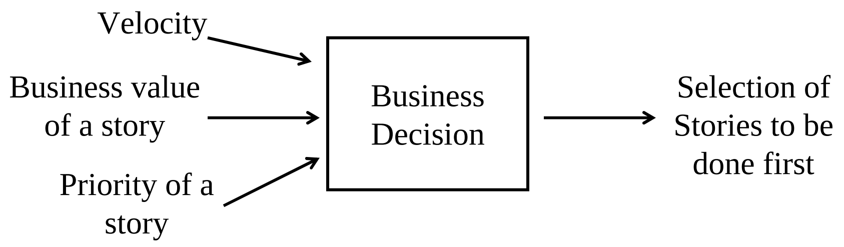
Development Process



Estimation

- Accurate estimation is hard
- Estimation comes from[6]
  - 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

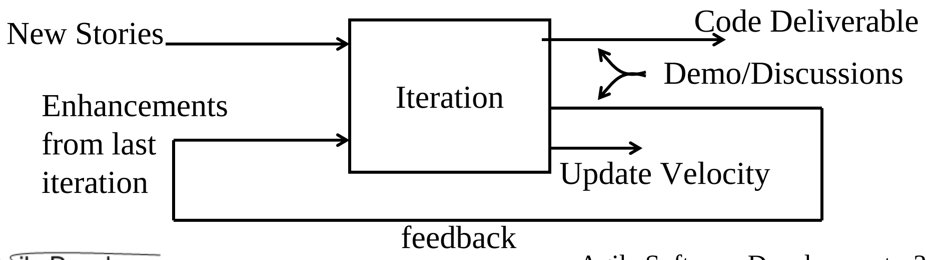
Release Planning


Velocity
Business value of a story
Priority of a story
Business Decision
Selection of Stories to be done first

- 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

- State of Software Development
- Agility
- Planning
- **Daily Activity**
- Conclusion

Standup meetings

- What's going on in your team?
- Do you waste a lot of time in meetings?
- Do you feel developers are isolated?

- Save time, help communicate – stand up for the meeting[7]
  - What did I do yesterday?
  - What's my plan for the day?
  - What's in my way?

Tell, don't wait to be asked

- You have promised a task end of this week
- You announce in the demo meeting that you have not completed it
- You have just invited your boss to micromanage you

- Communicate the status when there is a change – don't wait to be asked
- Information Radiators – techniques for keeping others informed[8]

Why Unit Test?

- Your code works
- You find the need to evolve your design
- You modify your code, you read through it, looks reasonable (and it compiled ☺), so you check it in
- What happens next?
- May be nothing for a few days or a week
- Then you hear a boom
- You curiously lean over to find your colleague in torn shirts, surrounded by smoke, uttering “I check out your code and it blue up”
- If you code sucks, wouldn’t you want to hear that from your computer than your colleague?

Unit Testing Benefits

- Unit Testing is an act of design than an act of verification[6,9,10]
- It helps provide instant feedback when code is changed
  - Substantially improves robustness of app
- Works as a great form of documentation
- Safety net when refactoring code

Test on all platforms supported

- "It works on my machine!" isn't good enough
- Do you promise to support multiple version of VM, different versions of OS, etc. ?
- It is your responsibility to make sure your code works on each
- But who has the time to test all that, we're already under pressure
  - Continuous integration can help on this

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[11]
- 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

Why refactor?

- Code that smells
- Code tends to rot over time
- If it is hard to understand, it will get hard to maintain

- You owe it to others to keep the code understandable and easier to maintain
  - Avoid duplication[15]

What is needed for Refactoring?

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

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[12, 13]
- 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

Setting up CI

- It takes a few hours to a day to set this up
- Cruise Control, Ant Hill, Damage Control, etc.[14]
- Benefit is enormous
- Figure out ways you can use this
  - No limits to your creativity

Quiz Time

Conclusion

- We all want to write software successfully
- Only constant is change
- How to keep up with it?
  - Communicate
  - Iterate
  - Unit Test
  - refactor
  - Integrate early and often
- Let’s succeed in development

References

1. "Software Project Management Practices: Failure Versus Success," Capers Jones (http://www.stsc.hill.af.mil/crosstalk/2004/10/0410Jones.html)
2. "Agile Software Development," Alister Cockburn, Addison-Wesley.
3. "Agile and Iterative Development: A Manager's Guide," Craig Larman, Addison-Wesley.
4. "Iterative and Incremental Development: A Brief History," Craig Larman, IEEE Computer, June 2003.
5. "Planning Extreme Programming," Kent Beck, Martin Fowler, Addison-Wesley.
6. "Agile Software Development, Principles, Patterns, and Practices," by Robert C. Martin, Prentice Hall.
7. "Agile Software Development with SCRUM," Ken Schwaber, Mike Beedle, Prentice Hall.
8. "Information Radiator," http://c2.com/cgi-bin/wiki?InformationRadiator.
9. "Test Driven Development: By Example," Kent Beck, Addison-Wesley.
10. "Pragmatic Unit Testing in Java with JUnit," Andy Hunt, Dave Thomas, Pragmatic Programmers.
11. "Refactoring: Improving the Design of Existing Code," Martin Fowler, Kent Beck, John Brant, William Opdyke, Don Roberts, Addison-Wesley.
12. "Continuous Integration," Martin Fowler, Matthew Foemmel, http://www.martinfowler.com/articles/continuousIntegration.html.
13. "Pragmatic Project Automation: How to Build, Deploy, and Monitor Java Apps," Mike Clark, Pragmatic Programmers.
14. "Continuous Integration Server Feature Matrix," http://docs.codehaus.org/display/DAMAGECONTROL/Continuous+Integration+Server+Feature+Matrix.
15. "The Pragmatic Programmer: From Journeyman to Master," Andrew Hunt, David Thomas, Addison-Wesley.
16. Some interesting articles to read - http://tinyurl.com/drnor

Download Slides from

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

Thanks!

Please fill out your evaluations!