# Some ideas behind Agile

Sriram Narayan
ThoughtWorks

## The Agile Manifesto

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- Customer collaboration over contract negotiation
- Responding to change over following a plan

ThoughtWorks

# Objectives

- Share a set of beliefs widely held to be true in the Agile community
- Establish 'common ground' for further argument.

ThoughtWorks

# An Aside - Fast feedback

- In a dynamic environment, fast feedback (short feedback cycles) is more valuable than getting things right first time.
- Minimizing 'Exposure to Downside Risk'
  - 1 unit of cost * 50% chance of failure = 0.5
  - 5 units of cost * 20% chance of failure = 1.0
  - Courage

ThoughtWorks

# 1. Coding Revisited

- More of a thought-activity than a typing activity
- Disambiguation of design
  - Requirements answers 'What?'
  - Design answers 'How?'
  - Coding answers: 'How exactly?'

ThoughtWorks

# A new look at Coding

| Car | Building | Software App |
|---|---|---|
| Market Research | Plan | Rqmnts and Planning |
| Design & Prototype | Design & Validate | Design & Code |
| Manufacture | Construct | Build |
| Test | Inspect | Test |
| Final Product =Car | Final Product =Building | Final Product =Executable App |

ThoughtWorks

# Programming != Manufacturing || Construction

- Better analogies for the activity of manufacturing or construction
  - Compilation of source code into executable
  - Burning CDs to ship the executable.
- Therefore, metrics like 'defect-rate' are of questionable value when applied to programming

ThoughtWorks

# Dev Team Org

- It is unproductive to separate design and programming roles within a team
- Necessitates document generation and lengthens the feedback cycle.

ThoughtWorks

# No one is above coding

- No ivory towers for Architects
- Its important to listen to/solicit feedback from the implementation work in progress.
- Conferring authority via designation makes the feedback part optional.

ThoughtWorks

# Code as a form of documentation

- Unit Tests when written as 'units' are an effective form of documentation in addition to providing 'insurance cover'
- Expressive code (self-describing code)
- Viewed as above, XP teams produce more documentation than traditional teams

ThoughtWorks

# To Summarize...

- Code is Design
- Everybody involved in development – be it programmer, designer or architect – should use feedback from the code being written for further work.
- Code is Documentation

ThoughtWorks

# 2. Documentation is an intermediate product

- The fundamental issue is communication
- Docs need time, skill and ongoing care
- Need to balance cost and benefit
- Different types of docs have different costs and benefits
- Within a project - Person to person communication is often more effective

ThoughtWorks

# Who decides how much?

- How much to invest in documentation?
- Let the stakeholder decide
- Not by precedent
- Not by dictates of a process

ThoughtWorks

# Documents of questionable value

- Minutes of technical meetings
- Traceability matrix
- Most plans (not planning)
- Mandating comments in code
- SQAs often insist on seeing evidence/documentation but rarely think about its quality.

ThoughtWorks

# Documentation tips

- Three useful questions
  - Cost of producing?
  - Cost of maintaining?
  - Cost of NOT producing?
- Who's the target? – docs meant for developers can very well be code
- Collaborative documentation e.g. wikis

ThoughtWorks

# To Summarize...

- Just enough documentation
- 'Just in Time' documentation
- What's right for your customer and your situation?
- Eliminating documentation by fostering communication within team is a form of disintermediation

ThoughtWorks
The art of heavy lifting

# 3. People dependence

- Typical reason: "No one should be indispensable – hence document"
- Person (singular) dependence may be risky
- But people dependence (plural) is okay and actually unavoidable for a knowledge organization
- Transferring knowledge from one person to a group is all about fostering communication.

ThoughtWorks
The art of heavy lifting

# Moving to 'People dependence'

- Pair programming and rotation
  - Instantaneous and continuous code review
  - Transfer of 'implicit' knowledge
  - Fosters 'collective ownership'
  - Reduces risk of 'person' dependence
- Unit Tests (as documentation)

ThoughtWorks

# To Summarize...

- Practices like 'pair programming' and 'unit testing' help you mitigate the risks of 'person dependence' without the overhead of unnecessary documentation.

ThoughtWorks

# Questions? Counter opinions?