Agile Fundamentals

A Course from ThoughtWorks Studios

ThoughtWorks studios

Key Concepts from Agile Fundamentals

Origins of Agile

- Agile is evolutionary
- Agile addresses wasted time, lost information, confusion, and miscommunication
- We cannot know everything before we start, but in Waterfall we try
- Project plans are an attempt to represent our efforts at knowing everything up front
- Waterfall comes from manufacturing, construction, and engineering
- NASA is represented as a paragon of effectiveness, with fewer defects than industry by three orders of magnitude (5 defects per KLOC vs. .0004 defects per KLOC = 1000x), but...
- NASA's costs are 170x industry standard (US$850/LOC versus US$5/LOC)
    - for a 100KLOC application, industry cost is US$500,000 while NASA cost is US$85,000,000
- Agile evolved from existing practices
- Agile Manifesto authored in February 2001[i]
- Agile is not a single thing, but a way of thinking and working
    - Kent Beck - XP
    - Ron Jeffries - XP
    - Ken Schwaber - Scrum
    - Jeff Sutherland - Scrum
    - Arie van Bennekum – DSDM
    - Alistair Cockburn – Crystal
    - Ward Cunningham – Wiki, FIT
    - Martin Fowler – Analysis patterns and many others
    - James Grenning – Planning poker
    - Jim Highsmith – Agile project management
    - Andrew Hunt – Pragmatic Programmers
    - Dave Thomas – Pragmatic Programmers
    - Jon Kern – model driven architecture, The Coad Letter
    - Brian Marick – agile testing
    - Robert Martin – Agile software development principles and patterns
    - Steve Mellor – Schlaer-Mellor method
- Some core practices:
    - Collocated teams
    - Pair Programming
    - Stand-Ups

- Commonalities between schools of thought:

Practice/Approach	Scrum	XP	Lean
Collocation	x	x	x
Collaboration	x	x	x
Information Radiators	x	x	x
Developer Practices		x	
Project Management	x	x	
Management Visibility	x		
Team Communication	x	x	x
Urgency			x
Cycle Time			x
Flow			x
Waste			x

The Agile Manifesto

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals.
Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development.
The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

- Waterfall and Big Design Up Front (BDUF)...
    - Assumes we can know everything up front - fallacy
    - Tends to be inflexible
    - Leads to detailed project plans
    - Reduces ongoing communication
    - Assumes high levels of trust, before trust is earned
    - Filled with shoulds, shalls, and musts
    - Written from the standpoint of the system, not the users

- Agile is...
    - User stories - told from the standpoint of the user and in terms of value
    - Collaboration and Communication - team and customer/user talking to each other continuously
    - Iterative - small, contained bodies of work, repeating the same steps each time
    - Incremental - delivering more value each Iteration
    - Continuous Feedback - communication between team and customer, ensuring that the value is being delivered
    - Delivering working product at the end of each iteration
    - Being adaptable - accepting changes from the customer without resistance based on rigid conformance to specs/plan

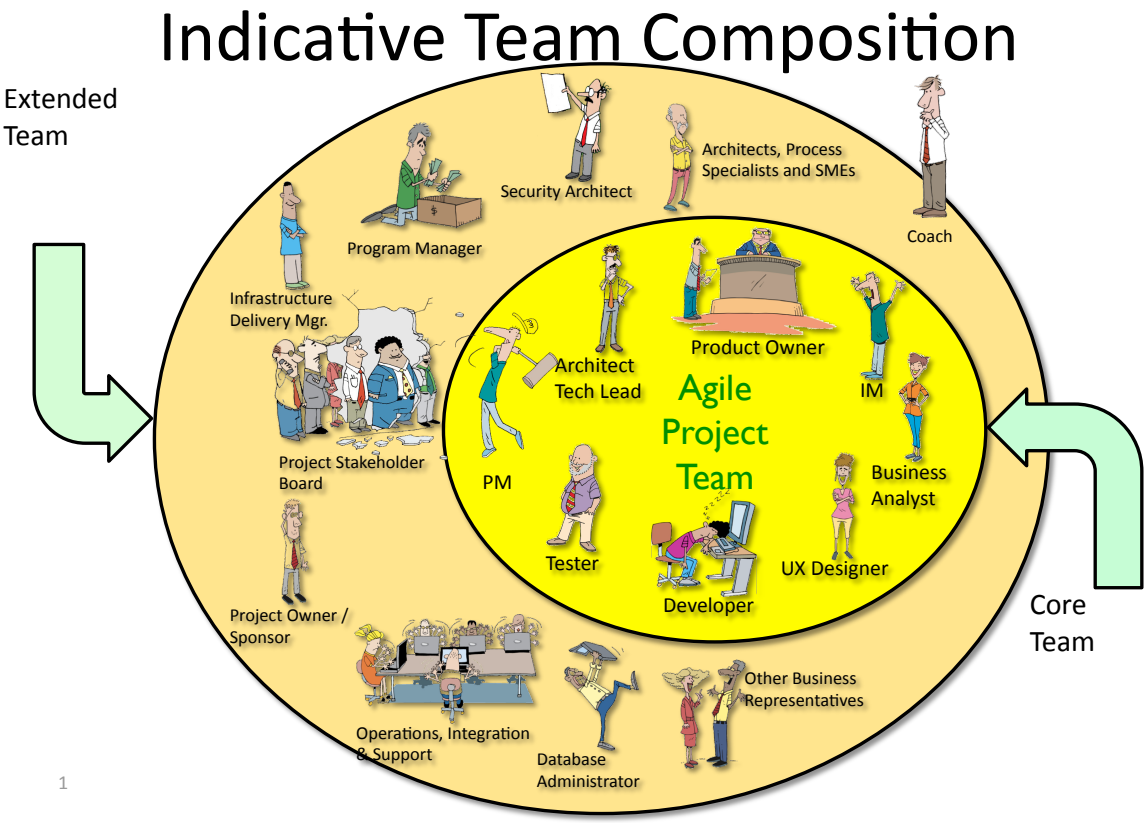
- Agile best practices...
    - Organizational
        - Automated build/deploy
        - Automated testing
        - Coding Standards
        - Continuous integration
        - Short releases
        - Collective ownership
        - Co-location
        - On-site customer
        - Adaptive planning
    - Team
        - Daily standups
        - Iterations
        - Sustainable pace
        - User stories
        - Velocity metrics
        - Card Wall
        - Pair programming
        - IKO/IPM
        - Transparency
        - Story based analysis
    - Individual
        - TDD
        - Simple design
        - Refactoring

- Conversation

- Agile Best Practices to Product Challenges

		Agile Best Practices																		
		Collaboration	Story based analysis	Last responsible moment	Adaptive planning	Value driven prioritization	Frequent releases	Simplicity	Refactoring	Collective code ownership	Continuous integration	Pair programming	Test driven development	Velocity metrics	Co-located teams	On-site customer	Information radiators	Showcases	Retrospectives	Daily stand ups
Project Challenges	Process inhibitors or distractions													x	x	x			x	x
	Siloed knowledge of the work or solution	x													x	x			x	x
	Changing requirements	x		x	x	x			x						x	x		x		
	Unclear and changing priorities					x											x	x		
	Mismatch between solution & business need	x			x	x	x									x		x		
	Late delivery dates	x	x		x		x	x						x			x			
	Cost of change				x	x	x	x			x	x	x				x			
	Budget pressure	x	x		x	x	x	x												
	Accurate visibility into project status/progress	x	x		x	x	x							x			x	x		
	Length of time to see ROI		x		x	x	x	x												
	Time to market		x			x	x	x												
	Changing technical environment									x	x	x	x							
	Evolving development tools								x	x		x	x							
	Legacy code without SMEs									x		x	x							
	Complex technical environments							x	x	x		x								
	High defect count/low code quality							x		x	x	x	x							
	Uncertainty/fear of changes							x		x	x	x	x							
	Lack of credibility with business and executives		x	x	x	x	x	x									x	x		

The Team



ThoughtWorks studios

The Responsibilities

Project Manager

The Agile PM reports progress, removes obstacles, manages budgets, and generally deals with outward facing project tasks.

Iteration Manager/Scrum Master

- Facilitates collaborative activity
- Encourages and mentors
- Carries water
- Removes boulders

Business Analyst

- Facilitates workshops
- Understands the business needs
- Focuses on business value
- Creates artifacts
- Supports developers
- Works with testers

User Experience Analyst

- Facilitates workshops
- Defines usability
- Conducts usability testing
- Usability =
  - Learnability
  - Efficiency
  - Memorability
  - Errors
  - Satisfaction

Quality Assurance Analyst – Tester

- Defines 'Done'
- Creates test plans
- Performs various tests
- Updates quality metrics

Architect/Technical Lead

The Architect is a working member of the team. The Architect has overall responsibility for envisioning the architecture of the system, and works side-by-side with developers and testers as they articulate the architecture through code and acceptance criteria/tests.

Developer

- Creates working software
- Adheres to coding standards
- Practices TDD
- Collaborates with BA and Testers

Product Owner/Customer/Business

- Provides vision for the solution
- Single point of escalation for prioritization

Business Stakeholders

- Represents a particular area of the business
- Participates in key activities
- Responds promptly

Implementation Stakeholders

- Represents areas of the delivery team
- Participates in key activities
- Responds promptly

Development Practices

Disciplines required to successfully develop and deliver

▪ Test-Driven Development (aka Test-Driven Design) (aka TDD)

- Cyclical
  - Articulate goal
  - Write test
  - Run test - FAIL
  - Write code
  - Run test - PASS
  - Refactor
  - Repeat

▪ Pair Programming

- Two brains better than one
- One task, two sets of hands and eyes
- Driver and Navigator
  - Switching roles
  - Switching pairs
- Real-time code review
- Fewer defects

▪ Smells

- Signals that something is not right
  - Subtle
  - Not the obvious
  - Indication of something deeper
- Categories of smells
  - Code smells
  - Design smells
  - Architecture smells
  - Team smells
  - Process smells

▪ Refactoring

- Part of a developer's job every day on every task
- Improve code without changing overall results
- No bug fixing
- No new functionality
- Remove unnecessary / unused code

Generalization & Specialization

- Specialization, while implying greater depth of knowledge and skill, can create resource shortages and bottlenecks
- Generalization, while implying shallower depth of knowledge and skill, creates greater resource availability and capability on the team
- Specialization is necessary to deal with some cases - specialists must be available
- Teams should be composed of generalists who can deal with architecture, design, development, test, continuous integration, business analysis,...
- "Specialists" on teams should be roles filled by generalists
- The more the knowledge and skill is shared among team members, the more capable the team becomes
- Generalization leads to greater collaboration and communication, and vice versa

Integration and Continuous Integration

- “Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.” *Martin Fowler*
- Uncover coding and integration problems early
- Early feedback
- Always have working software
- Break the build, fix the build

Quality

- Quality is subjective
- Quality is judged by the customer/consumer
- Quality is continuous and part of everything

Key Activities

- Gathering Acceptance Criteria
- Clarifying Stories
- Assure Quality

Types of Testing

Types are not exclusive - Acceptance Testing may include Functional Testing, for instance.

- Unit Testing / TDD
- Integration Testing / Continuous Integration (CI)
- Functional Testing
- Acceptance Testing
- Regression Testing
- Automated Testing
- User Acceptance Testing (UAT)

Working with Defects/Issues and Risks

- Defects are "stories"
- Defects are processed the same way as other stories, except...
  - ...when production is affected and the defect is critical
- Defects generally share the same resources as development and test
- Defects are visible in the same way as other stories

- Issues are present/past, while Risks are future-looking

- Maintain a risk log
- Risks should have attributes
  - likelihood
  - criticality
  - mitigation strategy (if any) - what to do when risk manifests
  - if no mitigation possible, be prepared

Stories and their elements

- Story - Three C's: Card, Conversation, Confirmation [ii]
- Story - INVEST[iii] acronym: Independent, Negotiable, Valuable, Estimatable, Small, Testable
- Parts of a story narrative:
  - Unique ID
  - Title
  - Description – Role, goal, value
  - Priority
  - Estimate
  - Analysis risk factors
  - Acceptance criteria
  - Assumptions
- Description:
  - As a <Role
  - I want to <Goal>
  - So that <Value>
- Acceptance Criteria:
  - Inception level – I'll know this is complete when….
  - Iteration level - Given, When, Then (multiples of these)[iv]

Tools to elicit stories

- Roles: sets of activities/permissions, not people and not personas
- Personas: created people used to check and inform design and implementation choices; used to create scenarios
- Scenarios: Persona filling a Role performing Activities
- Prototyping: Paper Prototyping - paper and pencil. Low cost approach to walking customer through the system. Through scenarios.
- Story mapping[v] – a way to represent stories into Activities, tasks to better understand what is being built.

- Contextual inquiry[vi] – Part of the Contextual Design methodology. In essence, observing people doing their jobs in the environment in which they typically do them in order to learn about subtle details of requirements that they might otherwise not mention.[vii]

Estimation & Release Planning

- **Accuracy and Precision**
  - Accuracy: hitting what you're aiming at, and being able to measure/prove it
  - Precision: hitting the same spot repeatedly
  - Agile teams, when estimating with NUTs, aim for precision, across team members and over time
- **Estimating**
  - NUTs - Nebulous Units of Time [viii]
    - Fibonacci Sequence
    - Integers, Powers of Two

        - T-Shirt Sizing
    - Ideal Time versus Calendar Time
- **Velocity** - the amount of work a team can complete in an Iteration
    - Raw Velocity Exercise - estimating team velocity before the team has any experience
    - Velocity calculation - average the last three Iterations
- **Prioritizing**: The Principle of Relative Priority: *All the decisions about item priority have to involve other items which are being traded in favor of the most important one(s). There is no absolute priority, only relative to other items.*
- **Span planning** – a technique to horizontally slice out releases. Using the story map from above, and prioritizing the story cards, the team can horizontally slice a release.

Managing the work

- **Planning Meetings**
    - Release Planning
        - All invested parties involved
        - First level of estimation
    - Iteration Planning
        - Pigs (core team) participate
        - Detailed information, estimation, tasks, acceptance criteria
- **Daily Standup / Daily Scrum**
    - Only Pigs participate
    - Chickens may observe, with permission
    - Answer three questions:
        - What did you do yesterday?
        - What will you do today?
        - What impediments/obstacles/blockers are preventing you from succeeding?
    - Done standing up, short, *not* a status report
- **Retrospectives**
    - Learn from the past
    - Plan for the future
    - Open communication channels
    - Improve team functioning
- **Inter team communications**
    - Project manager connects with all leads
        - Iteration manager, if applicable
        - Business analysis
        - Tech
        - Product Management
        - Escalation
        - Testing
- **Cross team communications**
    - Each discipline will have a cross team huddle

Visibility and Communications

- **Information Radiators**
    - Static sources of information
    - Always updated

    - No one should have to ask for status
- **Card wall**
    - Virtual or physical cards
    - Shows a single iteration flow
    - Shows the state of a card in the iteration
    - Analysis stage is typically referring to next iteration stories
    - Testing stage may be referring to previous iteration stories
- **Daily Stand-Up** [ix]
    - Only Pigs participate
    - Chickens may observe, with permission
    - Answer three questions:
        - What did you do yesterday?
        - What will you do today?
        - What impediments/obstacles/blockers are preventing you from succeeding?
    - Done standing up, short, *not* a status report
- **Burn-Down Chart**
    - Starts with scope, reflects completion of work towards "no work left to do"
    - Could be at the iteration or release level
- **Burn-Up Chart**
    - Scope versus work completed, gives trend line, predicts completion
    - Always accurate *at the moment it is being viewed*
    - Accurate prediction *given what we know today/now*
- **Showcases**
    - Sharing *what has been accomplished in this Iteration*
    - Only show completed stories - "done"
- **Risk Log**
    - No difference in tracking and managing risk in an agile environment
    - The difference is that there is much more opportunity to identify risk.

Obstacles

Miscommunication

- Between teams
- Within teams, between roles
- Between stakeholders and Teams

Lack of Understanding

- Folks who aren't involved in the day to day don't get the speed at which we will now work
- Folks need to understand why?
  - Why collocation
  - Why customer collaboration
  - Why pairing
  - Why TDD

Mixed Support

- Who is driving Agile in the organization?
  - From the top
  - Grass roots
- Support for the learning and the initial chaos
- Support for dedicated resources

Critical Elements of Agile Adoption

Vertical commitment

- Executives should lead by example
- Expect and accept that things will get slower at first
- It's OK to try new things and potentially fall short

Clear Understanding

- What do we expect to achieve as an organization using Agile methods?
- What types of mis-steps are considered acceptable?
- What benefits do we expect to gain as a team?
- What is agile and how does it work?

Common Language

- Roles and responsibilities
- Qualities of a story
- Estimation scale

- Business domain
- Application domain

Collocation

- Enhances communication
- Distributed teams are challenged and will experience a different level of adoption
- Immediate clarity on a question
- Osmotic learning

Adaptive planning

- Adapt the plan based on new information
- New information is potentially gained each iteration/scrum

Coaching

- Experiential support through the chaos

Creating a Roadmap

Critical questions

•What practices can we adopt and which do we need to put aside?

•When is specialization ok?

•How can I justify pairing when it is two bodies working on the same thing?

•What if our people do not want to do the engineering practices?

From Assessment to Execution

- Refine understanding of current state
- Identify ideal target state based on goals and constraints
- Form plan to go from current to target state
- Execute plan with periodic reviews

Links and References

Web Sites

Agile Manifesto: http://www.agilemanifesto.org

Alistair Cockburn/Crystal: http://alistair.cockburn.us/Crystal+methodologies+main+foyer

Kent Beck/Extreme Programming: http://www.threeriversinstitute.org/

Ken Schwaber: http://www.controlchaos.com

Jeff Sutherland: http://jeffsutherland.com

Mike Cohn: http://www.mountaingoatsoftware.com/

Mary & Tom Poppendieck/Lean: http://www.poppendieck.com/

General Resource: http://www.InfoQ.com

Good survey/summary of Agile: http://en.wikipedia.org/wiki/Agile_software_development

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

About Test-Driven Development: http://en.wikipedia.org/wiki/Test-driven_development

About Pair Programming: http://en.wikipedia.org/wiki/Pair_programming

About Refactoring: http://en.wikipedia.org/wiki/Refactoring

About Code Smells: http://en.wikipedia.org/wiki/Code_smell

Code Smells and Refactoring: http://c2.com/xp/CodeSmell.html

Agile Testing: http://testobsessed.com/wordpress/wp-content/uploads/2008/08/AgileTestingOverview.pdf

Low-Fidelity Prototyping: http://www.usabilityfirst.com/glossary/term_378.txl

Low-Fidelity Prototyping: http://interfacematters.com/2007/05/low-down-on-low-fidelity-prototyping.html

Flow Diagrams: http://www.agilemodeling.com/artifacts/uiFlowDiagram.htm

Contextual Design: http://en.wikipedia.org/wiki/Contextual_design

Contextual Inquiry: http://en.wikipedia.org/wiki/Contextual_inquiry

- **Story Mapping/Span Planning**

Jeff Patton on Story Mapping: http://www.agileproductdesign.com/blog/the_new_backlog.html

Alistair Cockburn on the Walking Skeleton: http://alistair.cockburn.us/Walking+skeleton

- **Information Radiators**

Alistair Cockburn on Burn Charts: http://alistair.cockburn.us/Earned-value+and+burn+charts

More on Burn Charts: http://guidewiredevelopment.wordpress.com/2009/01/29/burn-up-and-burn-down-charts

ThoughtWorks Studios on Card Walls: http://studios.thoughtworks.com/mingle-agile-project-management/2.3/help/card_rankings.html

- **Estimating**

http://en.wikipedia.org/wiki/Accuracy_and_precision

http://www.c2.com/cgi/wiki?IdealProgrammingTime

Books/Non-Web Reading:

- About Face 3: The Essentials of Interaction Design by Alan Cooper
- Agile Estimating and Planning by Mike Cohn
- Agile Retrospectives: Making Good Teams Great by Esther Derby
- Agile Software Development With Scrum by Ken Schwaber and Mike Beedle
- The Art of Agile Development by James Shore and Shane Warden
- Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series) by Robert C. Martin
- Crystal Clear by Alistair Cockburn
- Continuous Integration: Improving Software Quality and Reducing Risk (Addison-Wesley Signature Series) by Paul Duvall, Steve Matyas, and Andrew Glover
- Doc's Selected Reading Recommendations: http://astore.amazon.com/anotherthough-20?_encoding=UTF8&node=5
- Domain Driven Design by Eric Evans
- Extreme Programming Explained: Embrace Change (2nd Edition) (XP Series) by Kent Beck and Cynthia Andres
- Extreme Programming Installed by Ron Jeffries and Chet Hendrickson
- Inspired by Marty Cagan
- Lean-Agile Software Development: Achieving Enterprise Agility by Alan Shalloway
- Lean Software Development: An Agile Toolkit by Mary Poppendieck
- Planning Extreme Programming by Kent Beck and Martin Fowler
- Recipes for Continuous Database Integration: Evolutionary Database Development by Pramod Sadalage
- Refactoring: Improving the Design of Existing Code (Addison-Wesley Object Technology Series) by Martin Fowler, Kent Beck, John Brant, and William Opdyke
- Refactoring to Patterns by Joshua Kerievsky
- Test Driven Development: By Example (Addison-Wesley Signature Series) by Kent Beck
- Testing Extreme Programming by Lisa Crispin
- The ThoughtWorks Anthology: Essays on Software, Technology and Innovation
- User Stories Applied by Mike Cohn

End Notes

[i] http://agilemanifesto.org/
[ii] http://xprogramming.com/xpmag/expcardconversationconfirmation/
[iii] http://xp123.com/xplor/xp0308/index.shtml
[iv] http://behaviour-driven.org/
[v] http://www.agileproductdesign.com/presentations/user_story_mapping/index.html
[vi] http://en.wikipedia.org/wiki/Contextual_inquiry
[vii] http://www.usabilitynet.org/tools/contextualinquiry.htm
[viii] http://c2.com/cgi/wiki?NebulousUnitOfTime
[ix] http://martinfowler.com/articles/itsNotJustStandingUp.html