What we will do today

- Overview of agile methods
- Knowledge sharing in agile teams
  - Exercise
- Agile project management
- Agile approach to quality
- Empirical evidence on agile approaches
- Adoption of agile methods

Questions are always welcome!!!
Overview

- Why agile methods?
- Principles of agile methods
- A 10,000 meter view on agile methods
Software Processes

What do you need to do for successfully developing software systems?

Waterfall Model (Linear, Sequential)

http://en.wikipedia.org/wiki/Waterfall_Model
Taylorism

- Frederick Taylor
- "The Principles of Scientific Management" (1911)
  - Hierarchical leadership
  - Fixed, not fluid
  - Split locations for manufacturing and office work
  - Work became specialized with divisional labor
  - Product/outcome focused – not customer focused
  - Manufacturing-like repeatable process

Tayloristic Methods?

- We avoid using:
  - "traditional" methodologies
    - since traditions change
  - "plan-driven"
    - since agile methods are also plan-driven
  - "heavy-weight"
    - since it implies that agile methods are "light-weight", which may add a negative connotation
  - "task-based"
    - since this points to the side effect of a method, rather than the cause
  - "rigorous"
- We prefer “Tayloristic”
Why Agile Methods?

- CHAOS Report, Standish Group
  - 66% of projects failed or are challenged in 2002
  - Large projects are failing more often than small projects
  - Only 52% of features make it into product

http://www.standishgroup.com/sample_research/chaos_1994_1.php

---

WE NEED THREE MORE PROGRAMMERS.

USE AGILE PROGRAMMING METHODS.

AGILE PROGRAMMING DOESN’T JUST MEAN DOING MORE WORK WITH FEWER PEOPLE.

FIND ME SOME WORDS THAT DO MEAN THAT AND ASK AGAIN.

© Scott Adams, Inc. Dist. by UFS, Inc.
Business perspective – the hype

- Better software quality
- More predictable deliveries
- Higher customer satisfaction
- Higher development productivity
- Lower software costs

Business perspective – the opposition

- It’s just hacking
- Refactoring and emerging design is costly
  - upfront work will save money
- Oral documentation is an oxymoron
- Design after first testing
- Software is never done ➔ costs ballooning
- Why should I pay two developers/consultants to do the job of one
- Where is the big picture?
- Users don’t know what they need
- We always did this – but in a waterfall kind of way
Overview

- Why agile methods?
- Principles of agile methods
- A 10,000 meter view on agile methods

Four Project Variables

- **Cost**

- **Scope**
  - Feature creep
  - Requirements churn

- **Time**
  - "Adding people to a late project just makes it later"
    Brooks, Mythical Man Month

- **Quality**
  - Disasters and software bugs
    
    http://www.zanger.informatik.tu-muenchen.de/persons/huckle/bugse.html
    http://www.mtholyoke.edu/~rzdalea/cs100/software_disasters/sd.htm
    http://www.csl.sri.com/users/neumann/illustrative.html
http://agilemanifesto.org/

**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.


---

**Agile Methods**

- Focus on business value
- Focus on delivering software not documentation
- Human centric: customer, developers
Business value

- What is business value?
- How to measure business value?
- How to decide that a project is a success?

Agile Core Strategies

- Iterative and Incremental Development
- Adaptive Project Management
- Collaborative, "whole team approach"
- Common shared vision and goals
Agile Practices

- Knowledge sharing ("verbal documentation", co-location)
- Fast feedback (on-site customer, continuous integration, short iterations, retrospectives)
- Emphasis on testing!!!!!!!!!!
- Emerging (simple) design & refactoring
- Pairing
- Business contracts

Agile versus tayloristic methods

- **Agile methods**
  - Human-centric
  - Tacit knowledge sharing
  - Code-centric
  - Replace documentation by face-to-face communication
  - Generalists
  - Plan and correct
  - Customer-focused

- **Tayloristic methods**
  - Process-centric
  - Explicit knowledge
  - Documentation-centric
  - Role specialization
  - Plan and control
  - Contract-focused
Learn To See & Eliminate Waste

- 7 wastes of manufacturing:
  - Inventory
  - Extra Processing
  - Overproduction
  - Waiting
  - Motion
  - Defects

- 7 wastes of software development:
  - Partially Done Work
  - Extra Processes
  - Extra Features
  - Waiting
  - Motion
  - Defects

Agile practices seek to eliminate waste

Overview

- Why agile methods?
- Principles of agile methods
- A 10,000 meter view on agile methods
  - Extreme programming
  - Scrum
  - Lean development
  - FDD
  - Agile modeling
  - Crystal
  - Adaptive software development
  - DSDM
  - ...
Extreme Programming

We're going to try something called Extreme Programming.

First, pick a partner. The two of you will work at one computer for forty hours a week.

The new system is a minute old and I already hate everyone.
Some Basic Facts

- Producing code is required to deliver a system
- Dollars spent on analysis and design are wasted if the system is never used
- Business requirements have to be the drivers for software development
- Requirements change

Extreme Programming (1/e)

Practices (2/e)

Scrum
Scrum Flow (Sutherland, Schwaber and Beedle)


Scrum: 15 min-daily meetings
Team members respond to basics:
- What did you do since last Scrum?
- Do you have any obstacles?
- What will you do before next meeting?

Sprint: 30 days
New functionality is demonstrated at end of Sprint

Features assigned to Sprint
Product Backlog
Selected Product Backlog

Sprint Backlog

Every 24 hours

Potential Shippable Functionality

Potentially Shippable Functionality

Lean Development

- Eliminate waste
  - Value stream mapping
- Amplify learning by feedback and frequent synchronization
- Deliver as fast as possible
- Empowerment: Self-determination, motivation, leadership, expertise

Vision: Anticipated ROI, Releases, Milestones
Crystal Structure

<table>
<thead>
<tr>
<th>Criticality</th>
<th>1 - 6</th>
<th>20</th>
<th>40</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life (L)</td>
<td>C6</td>
<td>C20</td>
<td>C40</td>
<td>C100</td>
<td>C200</td>
<td>C500</td>
<td>C1000</td>
</tr>
<tr>
<td>Essential money (E)</td>
<td>D6</td>
<td>D20</td>
<td>D40</td>
<td>D100</td>
<td>D200</td>
<td>D500</td>
<td>D1000</td>
</tr>
<tr>
<td>Discretionary money (D)</td>
<td>E6</td>
<td>E20</td>
<td>E40</td>
<td>E100</td>
<td>E200</td>
<td>E500</td>
<td>E1000</td>
</tr>
<tr>
<td>Comfort (C)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Prioritized for Legal Liability

Prioritized for Productivity & Tolerance

large military system

banking system

medium-sized productivity tool

small utilities

DSDM Process

Feasibility

Business Study

Agree schedule

Design and Build Iteration

Create Design Prototype

Review Design Prototype

Identify Design prototype

Agree Schedule

Create Design Prototype

Review Design Prototype

Identify Functional Prototype

Review Prototype

Identify Functional Prototype

Review Prototype

Agree schedule

Create Functional Prototype

Review Functional Prototype

Identify Functional Prototype

Review Prototype

Agree schedule

Create Functional Prototype

Review Functional Prototype

Identify Functional Prototype

Review Prototype

Agree schedule

Create Functional Prototype

Review Functional Prototype

Identify Functional Prototype

Review Prototype

Agree schedule

Create Functional Prototype

Review Functional Prototype

Identify Design prototype

Agree Schedule

Create Design Prototype

Review Design Prototype

Identify Design prototype

Agree Schedule

Create Design Prototype

Review Design Prototype

Identify Design prototype

Agree Schedule

Create Design Prototype

Review Design Prototype

Identify Design prototype

Agree Schedule

Create Design Prototype

Review Design Prototype

Identify Design prototype

Agree Schedule

Create Design Prototype

Review Design Prototype

Identify Design prototype

Agree Schedule

Create Design Prototype

Review Design Prototype

Identify Design prototype

Agree Schedule

Create Design Prototype

Review Design Prototype

Identify Design prototype

Agree Schedule

Create Design Prototype

Review Design Prototype

Identify Design prototype

Agree Schedule

Create Design Prototype

Review Design Prototype
Agile Software Processes: Highlights

- Increase responsiveness of software teams
  - Changing requirements
  - Strong customer involvement
- Reduce overhead to speed up development
- Focus on people, collaboration, communication
- Deliverable: potentially shippable product functionality
- Source-code centric
- Low overhead tools
- Empowering developers and clients
- Time-boxed iterations (2 weeks – 4 months)
- Fast feedback

Knowledge Sharing
Knowledge Sharing Exercise

- Split up into teams of approx. 6-8 people
- Each team splits itself up into analysts, messengers, developers
  - **Analysts**: describe requirements on paper
  - **Messengers**: carry notes from analysts to developers and back
    - are not allowed to communicate in any way with other groups
  - **Developers** implement specification
- **Task**: specify an example drawing so that the developers can reproduce it
- **Messages**: only text in usual left-to-right multi-line format, no drawings
- **Process**:
  - Self-organize (5 mins)
  - Execute task (20 mins)
  - Debrief (15 mins)

10% Communication Error

**Tayloristic way:**

\[ (90\%)^6 = 53\% \text{ info gets to developer} \]

**Agile way:**

\[ 90\% \text{ info gets to developer} \]
Knowledge Sharing: Tayloristic Way

- Knowledge-as-object
- Focus on explicit knowledge and externalization of knowledge in documents or repositories

Knowledge Sharing via Documentation

- Tayloristic way:

“document what you do” and “do what have been documented”
Documentation: The Tayloristic Way

Ideally:
- capturing knowledge gained in the activities of a software project lifecycle
- ensuring product and process conformance to prior plans
- supporting quality improvement initiatives
- satisfying legal regulations

In reality:
- often stays outdated
- good documentation but no clean, working software does not have a business value

Knowledge Sharing: Tayloristic Way

- Experience Factory
  - responsible for intra-team learning
- Retrospectives
Build It and They'll Come Fallacy

- Warehouse like approaches fail
- Experience factories become experience cemeteries
- Little interest and lack of contributions
- Common remedy – introduce an incentive system
  - a Dove bar
  - a bonus grade (participation grade)
  - "give to get" schemes …

Main problem: people focus on collecting and storing knowledge instead of on reusing it.

The Case of the Missing "IF"

- IF it is very difficult and expensive to gather, understand, and modify requirements,
  THEN we must establish formal requirements contracts with our stakeholders
- IF once software is written, it is very difficult to change,
  THEN we must establish detailed documentation, precisely describing the system requirements in all their splendid detail
- IF a substantial number of our software testers are very inexperienced,
  THEN we must document every test case in excruciating detail
- IF we are concerned that we will forget test cases because of their number & complexity,
  THEN we must document every test case in excruciating detail
- IF we are concerned about possible interactions between system components,
  THEN we must test all combinations of all components

IF you know what you are doing,
THEN do stuff
Documentation: The Agile Way

- Promote “lean and mean” documentation:
  - externalized to user stories (XP), models (FDD, AM)
    - made public
    - collectively owned
  - self-documenting designs
  - easy-to-read acceptance tests
  - self-describing code
  - self-describing unit tests
  - coding standards and guidelines (both industry-wide and internal)
  - Scrum: “work-in-progress” the only type (except for special docs for marketing, auditors, and user docs)
  - just-in-time, just-enough, just-because

Continuous Knowledge Sharing

- daily stand-up meetings (scrum)
- pairing
- pair-rotation
- collective code ownership
- on-site customer
- retrospectives (during +post mortem) e.g.
  - post-sprint meetings in Scrum,
  - reflection workshops in Crystal,
  - post-iteration phases in DSDM
- Wiki – open collaboration, open editing, exchange of ideas
Knowledge Sharing: Highlights

- Effective knowledge sharing is imperative to succeed
- The higher the level of abstraction, uncertainty and/or equivocality, the "richer" medium is needed
  - Conversations are the "richest"!
- Long knowledge transfer chains result in filtered/distorted messages
- Tayloristic methods:
  - Focus on explicit knowledge and externalisation of knowledge in documents or repositories
- Agile methods:
  - Leverage tacit knowledge through socialisation (communication + collaboration)

Agile Project Management
Four Project Variables

- **Cost**
  - CHAOS Reports, Standish Group, 1994-2002

- **Scope**
  - Feature creep
  - Requirements churn

- **Time**
  - “Adding people to a late project just makes it later”
    - Brooks, Mythical Man Month

- **Quality**
  - Disasters and software bugs
    - [http://www.mtholyoke.edu/~rzdalea/cs100/software_disasters/sd.htm](http://www.mtholyoke.edu/~rzdalea/cs100/software_disasters/sd.htm)

Agile Project Management

- Iterative & time boxed
- Project scope not fixed at beginning
  - reactive to changing business needs
- Business value focused to keep customers happy
- Team effort
Time Boxes

- Never slip a date → change scope
- Sometimes external deadlines are HARD

Advantages
- Increased motivation
  - Successful delivery keeps developers and customers happy
- Faster feedback
- Creates a constant project heartbeat
- Deadlines create pressure (counters: work fills time available)

Advantages of flexible dates
- Release only when required scope is completed
- Overly optimistic deadlines are made more realistic

Project Vision

- Vision = Statement of what the business will look like once the new system is implemented.
- Used to establish a project budget
- Established by product owner
  - Provides/finds funding for projects
- Vision includes
  - Anticipated benefits for business
  - Assessment criteria for management to evaluate progress and conformance to vision
    → Management oversight needed
A Matter of Trust: Business Contracts

- Fixed scope/fixed price contracts
  - Trust by contract
  - Attempts to move technical risk to development side
  - Contract requires documentation → imposes process
  - Opposing sides of table

- How are fixed prices derived by development organization?

- Time and expenses contracts: Fixed budget/variable scope/early termination
  - Trust by feedback and involvement
  - Collaborative environment
  - Changes easy
  - Issues:
    - No time limit on project
    - No guaranteed functionality

Some Terminology

- Project planning, iteration planning, planning game (XP), sprint planning (Scrum)
- Story card/index card (XP), backlog entry (Scrum), feature/feature set (FDD)
- Customer, goal donor/user, gold owner/client, product owner, scrum master
- Spike
Agile Project Planning

- Project vision → the really big picture
- Release planning → strategic picture
  - Chooses a few months worth of user stories/features
  - Date and scope
  - Can be changed
  - Creates product backlog
- Iteration planning → tactical picture
  - Few weeks
  - Set of stories prioritized by customer
  - Creates sprint backlog
  - Define set of tasks for each story
  - Task granularity: 1-3 work days → estimation accuracy

Managing a Release

- Value Driven Releases

- Business value = \( f(\text{cost, time, functionality, quality}) \)

- 80% of the business value can be derived from 20% of the functionality
- Linear development: Christmas wish lists
- Iterative development: prioritized wish list
Minimum Marketable Features

- Components with intrinsic marketable value
- Creates business value by
  - Competitive differentiation
  - Revenue generation
  - Cost Saving
  - Brand projection
  - Enhanced loyalty

M. Denne, J. Cleland-Huang: Software by Numbers, Prentise-Hall, 2004

---

Scrum Flow (Sutherland, Schwaber and Beedle)


Scrum: 15 min daily meetings
- What did you do since last Scrum?
- Do you have any obstacles?
- What will you do before next meeting?

Every 24 hours
- Daily Scrum
- Sprint 30 days
- Potentially Shippable Functionality

Features assigned to Sprint

New functionality to be demonstrated at end of sprint

Product Backlog: Emerging, prioritized requirements

Selected Product Backlog

Sprint Backlog

Visit:
- Annotated ROI
- Releases, Milestones

Tutorial F8
Agile Methods: Moving Towards the Mainstream of Software Industry
© 2006 Frank Maurer & Grigori Melnik
Agile Requirements Definition

- User stories/Backlog Entries
  - Feature requests
  - On index cards
  - Short descriptions of a feature
  - In customer language, no techno babble
  - Provide value to customer
  - Independent of each other
  - Testable
  - Small $\rightarrow$ decompose large stories
- Estimated by developers:
  - best case, most likely, worst case
- Collect story cards and prioritize them

When is a User Story Done?

- All unit tests pass
- All acceptance test pass
- The customer accepts it
- All refactorings are done
Who Decides?

- Business decisions
  - Scope: which "user stories" should be developed
  - Priority of stories
  - Composition of releases
  - Release dates

- Technical decisions
  - Time estimates for features/stories
  - Elaborate consequences of business decisions
  - Team organization and process
  - Scheduling

Estimation in XP

- Ideal Engineering Time (IET, Gummy Bears, Effort, …)
  - No interruption
  - "Complexity" or "size" of task
  - Estimate adds the hours of a pair:
    2 developers, 3h each ⇒ IET=6h

- Based on experiences from the past

- Team effort
  - Optimism wins
  - Team usually does not overrule the estimate of programmers responsible for a task

- Issue:
  Effort estimates done by developers might lead to slack
Scrum Study (Mann/Maurer)

- 2 year longitudinal case study
- Researcher embedded in development team
- Overall results:
  - Reduced overtime
  - Increased customer satisfaction

![Average Percent Overtime Worked By Team](chart)

Daily Scrums (Stand-up Meetings)

- Daily 15 minute status meeting
- Same place and time every day
- Meeting room
  - Chickens and pigs
- Three questions;
  - What have you done since last meeting?
  - What will you do before next meeting?
  - What is in your way?
- Impediments and
- Decisions

Based on Ken Schwaber’s Certified Scrum Master course
Reporting: Tracking Progress & Metrics

1. Two questions
   - How many hours/days have you worked?
   - How many more does it take?
   Which one is more important

- Project metrics
  - Actual time worked on a task
  - Work burndown graph
    - Per iteration
    - #Backlog → project
  - #Bugs
  - #Stories completed
  - #Acceptance tests defined and passing
  - #Unit tests
  - Test coverage
  - ...

---

Project Tracking: Work Burndown Charts

Based on Ken Schwaber’s Certified Scrum Master course

---

Underestimating

Overestimating
Agile Project Management - Summary

- Vision, release, iteration
- Short horizon for detailed planning
- Reporting needs to tie in with vision and business value
- Adaptive and flexible
- Team effort

Agile Approach to Quality
Tayloristic vs. Agile QA

- **Tayloristic**
  - separate, independent test teams
  - process groups
  - inspections/reviews
  - test plans
  - antagonism

- **Agile**
  - cross-functional teams (testers are part of development team)
    - still make use of independent testers
  - conformance to process is not a primary goal
  - pairing
  - automated suites
  - interaction and collaboration

Agile QA Practices

- Take a good practice and turn the knob up to the extreme:
  - code inspections
    - pair programming
  - test case reviews
    - test-driven development
  - unit testing
    - merciless, automated unit testing & continuous integration
  - retrospectives
    - do them during and after and frequently
Agile Quality Approach

Ux Designers

Customer Team + Testers

Development Team

Pair Programming

SYSTEM (UNDER TEST)

Pair Programming < Pairing
Testing – The State of the Practice in Agile Teams

It All Comes Down to Feedback!!!!

- Rapid
- Continuous/very frequent
- Comprehensive
- From all stakeholders (incl. the development team)
- From code as well
Testing – to the forefront of the development

- Write tests before production code
  - Unit tests → developer
  - Feature/acceptance tests → customer/QA
- Strong emphasis on regression testing
  - Unit tests need to execute all the time
  - Tests for completed features need to execute all the time
- Acceptance tests show progress on user stories
- Automate as much as possible

Acceptance (Customer) Tests

- Are used to judge if the product is acceptable to the customer
- Coarse grained tests of business operations (functional)
- Story-based (contain expectations)
- Contain
  - Happy paths (confirmatory)
  - Sad paths
  - Alternate path (deviance)
- Comprehensive, end-to-end
The Value of Concrete Examples

- Most customers find it difficult or impossible to express business rules and system requirements in the abstract.
- The following are often too abstract or too imprecise:
  - requirements specs
  - use cases
  - UML diagrams
  - formal specs in Z, etc.
- Trying to do this all upfront just makes it impossible:
  - few customers know completely where the true business value lies when they start
  - they need to evolve their understanding with feedback

The Value of Concrete Examples

- The development of examples enables the customer to express and refine how a system could be used:
  - getting clearer about the underlying business rules that exist (analysis)
  - getting clearer about how a system can be used to provide real business value (synthesis)
- Evolution and feedback are essential, at small and large scales:
  - at the story/feature level:
    - develop very simple examples and refine them to add complexity to get to executable acceptance tests
  - at the iteration level:
    - change and add to the existing suite(s) of acceptance tests to make the system richer
Customers "Write" the Tests

- Solidifies the meaning of user stories/features
- Shows the customer that their own needs are being met
- Demonstrates that actual value is coming from completed code
- Customer can pair with Interaction/Ux Designers and/or Testers

Executable Acceptance Testing

- FIT (Cunningham)
- Test cases are specified in tables
  - can be specified in Excel, HTML, Wiki...
- Parses tables
- Generates test cases using this data
- Automatically runs the test cases
- Shows the result on a Web site

<table>
<thead>
<tr>
<th>TEST RESULTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>400</td>
</tr>
</tbody>
</table>
How FIT Works

- **Customers/business analysts** provide test scenarios
- **Developers** write “fixtures” that make the tests run and eventually pass
- Fixtures are refactored into calls to clean working code

**Case: EasyTix**
Starts With a Conversation

- Warm communication
- One scenario at a time
- Let the user write it

Customer and Ux Designer

user searches for event U2

system displays search results
check event U2
check tickets available 10

user buys 1 ticket with card 352955

system returns eticket
check number of seats purchased 1
check event purchased U2

user searches for event U2

system displays search results
check event U2
check tickets available 10

user buys 1 ticket with card 6529554928334857

system returns eticket
check number of seats purchased 1
check event purchased U2
Write a FIT Test

FindAndBuySingleTicketForSpecificEvent

Make It Pass
Rinse and Repeat

And So On…

- Do the simplest thing possible to make it pass
- Then refactor into clean code that works
- Rinse and repeat
Fixtures Mediate Between Tables and SUT

Internationalization

- Full support of Unicode

<table>
<thead>
<tr>
<th>CalculateCreditRu</th>
<th>Описание</th>
<th>Стоимость</th>
<th>Количество</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>записная книжка</td>
<td>1120.25</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>печатающее устройство</td>
<td>399.00</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>бумага</td>
<td>5.00</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CalculateCredit</th>
<th>月数</th>
<th>信頼度</th>
<th>使用度</th>
<th>クレジットを許可する</th>
<th>クレジットの限度</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>真</td>
<td>5000.00</td>
<td>真</td>
<td>1000.00</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>真</td>
<td>0.00</td>
<td>偽</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>偽</td>
<td>0.00</td>
<td>偽</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>真</td>
<td>6000.00</td>
<td>偽</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>真</td>
<td>5500.00</td>
<td>真</td>
<td>1000.00</td>
<td></td>
</tr>
</tbody>
</table>
Test Table Styles

- Defining business processes/workflows (state changes)
  - DoFixture, ActionFixture
- Defining calculations and constraints
  - ColumnFixture, CalculateFixture
- Defining business objects within acceptance tests
  - SetUpFixture, ObjectFixture
- Defining given and expected lists and queries within storytests
  - RowFixture, SetFixture, ArrayFixture, etc
- Defining boards, layouts, custom table formats
  - TableFixture, GridFixture

Good Acceptance Test

- The story is motivating.
  - A stakeholder with influence would push to fix a program that failed this test.
- The story is credible.
  - It not only could happen in the real world; stakeholders would believe that something like it probably will happen.
- The story involves a complex use of the program or a complex environment or a complex set of data.
- The test results are easy to evaluate.
  - This is valuable for all tests, but is especially important for scenarios because they are complex.
Unit (Programmer) Tests

- Some executable acceptance testing leads to...
- Unit testing
  - more granular
  - lower level
  - testing of units (classes, methods)
  - automated (xUnit)

"Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead." — Martin Fowler

Unit Test Guidelines (Michael Feathers)

- A test is not a unit test if:
  - It talks to the database
  - It communicates across the network
  - It touches the file system
  - It can’t run correctly at the same time as any of your other unit tests
  - You have to do special things to your environment (such as editing config files) to run it.

- Tests that do these things aren’t bad. Often they are worth writing, and they can be written in a unit test harness. However, it is important to be able to separate them from true unit tests so that we can keep a set of tests that we can run fast whenever we make our changes.
Test-First Design (Test-Driven Development)

Rules of Engagement

- Test everything that can possibly break
- Tests come first
  - must fail at first as you don’t have any code yet
- All tests run at 100% all the time
- Eliminate duplication
TDD and Courage

- TDD is a way of managing
  this-is-a-hard-problem-and-I-can’t-see-the-end-from-the-beginning **fear** during programming

- Benefits
  - Thinking before doing
  - Automatically compares specification with implementation
  - Automated regression testing supported
  - Enables refactoring

Implications of TDD

- You must design organically, with running code providing feedback between decisions
- You must write your own tests, since you can’t wait twenty times a day for someone else to write a test
- Your development environment must provide rapid response to small changes
- Your designs must consist of many highly cohesive, loosely coupled components, just to make testing easy
TDD and Test Organizations

- Unit tests – natural by-product of TDD
  - useful enough to keep running as long as the system is running
  - don’t replace the other types of testing (performance, stress, usability…)
- If the defect density of test-driven code is low enough, then the role of professional testing changes:
  - “adult supervision” becomes an amplifier for communication

Testers Outside of Development Team

- Provide additional value (not driven by delivery but quality)
- Specialized services
  - security testing, usability testing, performance testing etc.
- Independent Validation & Verification when required by legal bodies, standards etc.
Resources

- fit.c2.com
- junit.org
- fitnesse.org
- agile-testing discussion group on yahoo.com
- Brian Marick's testing.com
- James Bach's satisfice.com
- Ron Jeffires xprogramming.com
- Other frameworks:
  - Acceptance: Exactor, TextTest, Ruby/Watir
  - Specialized unit: JWebUnit, Selenium

Empirical Evidence
The Issue …

Do agile methods work because of their engineering and management practices or because the people who introduce them are simply very good developers?

Agility in General – Hard Data Anyone?

- Too much of the Agile Practice is based on assumption and opinion
  - Little hard empirical data
  - Minimal focus on data collection
  - Data that does exist is passed on by “oral tradition”
  - Data that does exist is often incomplete or tainted
  - High-profile consultants provide anecdotal evidence

- This makes it difficult to know what works, which in turn makes it difficult to adopt new methodologies.
Existing Studies

- Only individual practices:
  - pair programming
  - distributed pair programming
  - test-driven development
  - acceptance testing
- Problems with external validity:
  - use students as subjects

Cost of pair programming

  - University study with 41 students
  - Higher quality code
    - Test cases passed individuals: 73.4%-78.1%
    - Test cases passed pairs: 86.4%-94.4%
  - Pairs completed assignments 40-50% faster (average 15% higher costs)
  - Pair programming preferred by students (85%)
Studies of TDD

- Effect on productivity and quality
- Controversial results:
  - Müller/Tichy, 2001 – positive effect
  - Müller/Hagner, 2002 – no significant effect
  - Maximilien/Williams, 2003 – positive effect
  - Erdogmus et al, 2003 – no significant direct effect
- But…TDD, at worst, increases the amount of test assets produced with no negative impact on productivity.

A Cross-Program Investigation of Students' Perceptions of Agile Methods (Melnik/Maurer)

- GOAL: to determine the perceptions of a broad student body on agile practices
- Online questionnaire
  - voluntarily and anonymously
- Total numbers of respondents: 240
  - out of 693 invited, 35% response ratio
- Over 3 years
  - 6 academic semesters
  - 7 courses

<table>
<thead>
<tr>
<th>Academic program</th>
<th>Semester(s)</th>
<th># of invitations sent out</th>
<th># of respondents</th>
<th>Response rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIPLOMA (2 years)</td>
<td>Fall 2001, Winter 2002</td>
<td>45</td>
<td>20</td>
<td>44%</td>
</tr>
<tr>
<td></td>
<td>Fall 2003</td>
<td>46</td>
<td>9</td>
<td>20%</td>
</tr>
<tr>
<td>POSTHORS (2+3 years)</td>
<td>Winter 2002</td>
<td>22</td>
<td>12</td>
<td>55%</td>
</tr>
<tr>
<td></td>
<td>Fall 2003</td>
<td>18</td>
<td>10</td>
<td>56%</td>
</tr>
<tr>
<td></td>
<td>Fall 2003</td>
<td>22</td>
<td>22</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Winter 2004</td>
<td>17</td>
<td>11</td>
<td>65%</td>
</tr>
<tr>
<td>JUNIOR (3 years)</td>
<td>Winter 2003</td>
<td>175</td>
<td>25</td>
<td>14%</td>
</tr>
<tr>
<td></td>
<td>Winter 2004</td>
<td>140</td>
<td>18</td>
<td>13%</td>
</tr>
<tr>
<td></td>
<td>Fall 2002</td>
<td>55</td>
<td>18</td>
<td>33%</td>
</tr>
<tr>
<td></td>
<td>Fall 2003</td>
<td>62</td>
<td>19</td>
<td>31%</td>
</tr>
<tr>
<td></td>
<td>Fall 2004</td>
<td>33</td>
<td>28</td>
<td>85%</td>
</tr>
<tr>
<td>SENIOR (4 years)</td>
<td>Winter 2003</td>
<td>24</td>
<td>16</td>
<td>67%</td>
</tr>
<tr>
<td></td>
<td>Winter 2004</td>
<td>14</td>
<td>9</td>
<td>64%</td>
</tr>
<tr>
<td></td>
<td>Fall 2002</td>
<td>11</td>
<td>8</td>
<td>73%</td>
</tr>
<tr>
<td></td>
<td>Fall 2003</td>
<td>18</td>
<td>17</td>
<td>94%</td>
</tr>
<tr>
<td>POSTDIPLOMA (2+2 years)</td>
<td>Fall 2003</td>
<td>240</td>
<td>240</td>
<td>100%</td>
</tr>
</tbody>
</table>
Questions Asked

- Did the students enjoy agile practices?
- What worked for them?
- What problems did they encounter?
- Whether they would use agile practices in the future (if allowed)?
- What were their impressions of individual practices?
- How did XP improve their learning?

Using XP Improves Productivity of Small Teams

“I believe that XP helps get more work done in less time and is very effective for small groups as it allows for the group members not to get stuck for extended periods of time.”
Using XP Improves Quality of Code

"Quality is built into the process (not a supporting concept but a core concept)."

Employee Satisfaction Study (Melnik/Maurer)

- Research question: Do agile methods lead to higher job satisfaction rates in software development teams than the average satisfaction in IT industry?
  - across different age groups and compensation levels
- Discover relationships between selected job satisfiers and the overall jobs satisfaction
- Comparative analysis
Comparative Analysis of Job Satisfaction (Melnik/Maurer)

- Web-based survey (459 complete, 286 partial responses)
- Some questions verbatim identical with Computerworld survey (936 responses)
- Overall job satisfaction by groups

<table>
<thead>
<tr>
<th></th>
<th>Agile</th>
<th>Non-agile (main study)</th>
<th>IT General (supplementary study)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall job satisfaction (%)</td>
<td>11.0%</td>
<td>7.6%</td>
<td>25.0%</td>
</tr>
<tr>
<td>IT General</td>
<td>21.4%</td>
<td>35.0%</td>
<td>31.3%</td>
</tr>
<tr>
<td>Non-agile</td>
<td>8.5%</td>
<td>11.0%</td>
<td>53.2%</td>
</tr>
<tr>
<td>Agile</td>
<td>29.8%</td>
<td>6.9%</td>
<td>29.4%</td>
</tr>
</tbody>
</table>

Job satisfiers

- Strong positive correlation
  - level of agile experience
  - ability to influence decisions that affect you
  - opportunity to work on interesting projects
  - relationships with users

- Statistically significant
  - workload satisfaction
  - opportunity for advancement
  - ability to influence day-to-day company's success

Satisfiers (as per Table 2)

<table>
<thead>
<tr>
<th>Satisfiers</th>
<th>Opportunity for advancement</th>
<th>Ability to influence decisions that affect you</th>
<th>Ability to influence day-to-day company's success</th>
<th>Opportunity to work on interesting projects</th>
<th>Salary</th>
<th>Connection between pay &amp; performance</th>
<th>Job security</th>
<th>Workload</th>
<th>Relationship with IT peers</th>
<th>Relations with users/customers</th>
</tr>
</thead>
<tbody>
<tr>
<td>χ², ρhos, ρhos, rel.</td>
<td>79.92, moderate</td>
<td>103.24, moderate</td>
<td>84.05, moderate</td>
<td>99.18, moderate</td>
<td>45.28</td>
<td>63.96, weak</td>
<td>64.34</td>
<td>67.42, weak</td>
<td>59.43, weak</td>
<td>88.82, moderate</td>
</tr>
<tr>
<td>Satisfiers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[|N=481, df=36, p<0.0001, Chi-square & Spearman’s correlation test|]
Agile Methods Adoption

Diffusion of Innovation (Rogers; Moore)
Adoption Status (Forrester Report, Nov 2005)

- Agile software development processes are in use at **14%** of North American and European enterprises.
- Another **19%** of enterprises are either interested in adopting agile or already planning to do so.

Adoption Status (Methods & Tools newsletter, May 2005)

At what stage is the agile approach (XP, Scrum, FDD, ...) adoption at your location?

- Not aware: 26%
- Not using: 16%
- Investigating: 14%
- Analysed and rejected: 3%
- Pilot projects: 4%
- Partial implementation (adoption of some agile practices): 17%
- Partial deployment (some projects are using this approach): 12%
- Deployed (all new projects are using this approach): 8%

Who Does Agile Today?

- ABB
- Bayerische Landesbank
- BMW Financial Services
- Boeing
- Borland
- Capital One
- Caterpillar
- CP Rail
- Credit Swiss Life, Inc.
- Daimler Chrysler
- First Union National Bank
- Ford Motor Company
- Great West Life
- IBM
- Lloyds Bank Insurance
- Microsoft
- Petro Canada
- Primavera
- Transcanada Pipelines
- UBS
- ... + hundreds of small/medium size companies
- ... + hundreds of consultants
- ... + dozens of user groups

Have Agile Methods Crossed the Chasm Yet?
Summary

Take-Away Points

- Agile practices are not new
- An alternative to Tayloristic "Design, Analyze, Implement, Litigate"
- Agile is a cultural transformation, attending to principles over practices
- No silver bullet
- Human-based methodologies
- Embrace change
- Cross-functional teams avoiding waste
- Empowerment of customer and developers
- Agile start with minimum process and introduce more as needed; Tayloristic start with maximum and you need to cut back to scale down
Where to Find Information

- Addison Wesley
  - XP series
  - Agile software development series

- North America: XP Universe/Agile Universe/Agile conference series
  - www.agile2006.org

- Europe: XP conference series
  - www.xp2006.org

- Canadian Agile Network
  - www.agilenetwork.ca

- Agile Alliance
  - www.agilealliance.org

- Scrum Alliance
  - www.scrumaillage.org

- Mailing lists at Yahoo and other places

References

- Ken Schwaber: Agile Project Management With Scrum, Microsoft Press, 2004
Thank You!

- Frank Maurer
  - maurer@cpsc.ucalgary.ca
- Grigori Melnik
  - melnik@cpsc.ucalgary.ca

- http://ebe.cpsc.ucalgary.ca/ebe