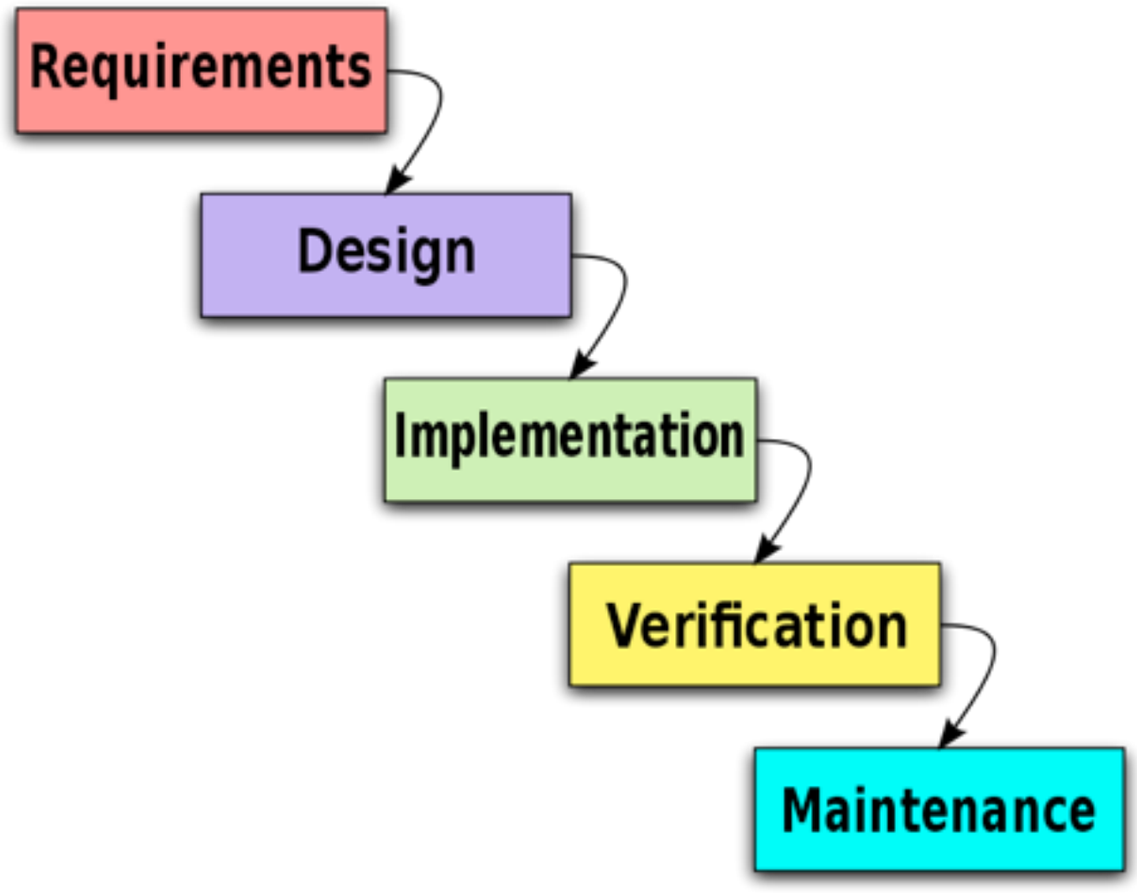
Software Engineering and Architecture
Introduction to Agile Software Development

Olivier Liechti
HEIG-VD
olivier.liechti@heig-vd.ch

MSE | MASTER OF SCIENCE IN ENGINEERING

“Traditional” methodologies: waterfall

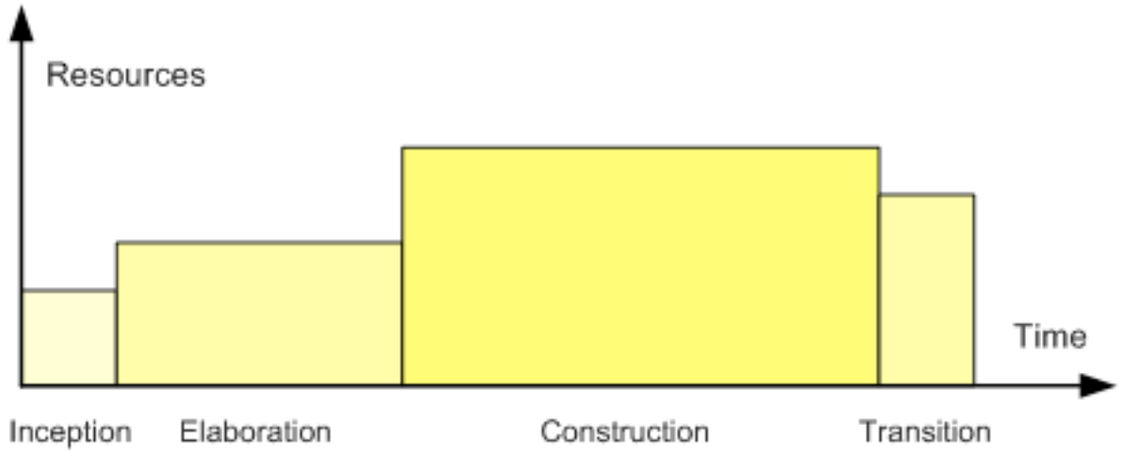
http://www.flickr.com/photos/44799719@N00/330191406/sizes/l/#cc_license

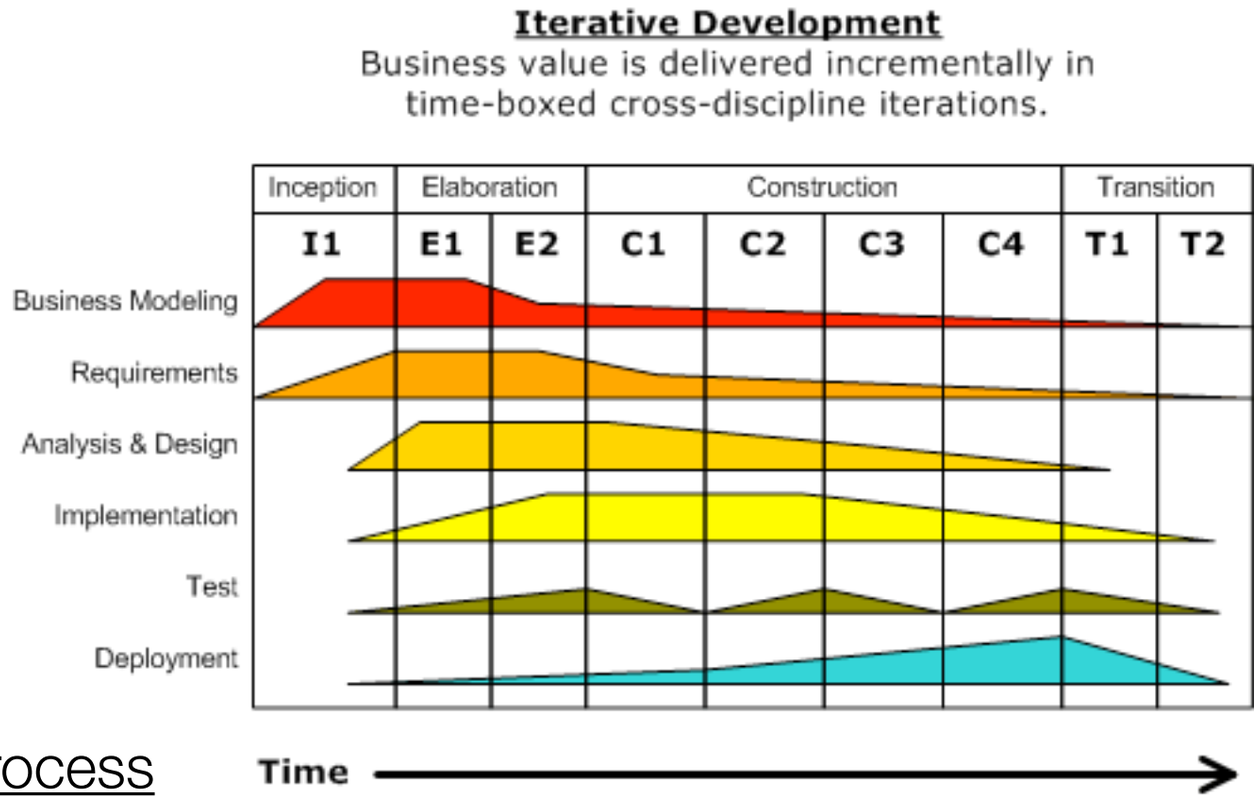


http://en.wikipedia.org/wiki/Waterfall_model

“Traditional” methodologies: Unified Process

- Principles
  - Iterative and incremental
  - Use case driven
  - Architecture centric
  - Risk focused
- Issues
  - Can be “process” and “artifacts” heavy.
  - Is there a “transition” phase?





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

Build software to create value

- **Let's assume that:**
  - The customer is able to completely specify requirements at the beginning of the project.
  - We are able to deliver a system that fulfills 100% of these requirements.
- **Question:**
  - Can we be sure that the project is successful?
  - Is there still a risk of failure?

The Agile Manifesto

- February 2001
- “Representatives from
  - Extreme Programming, SCRUM, DSDM,
  - Adaptive Software Development, Crystal
  - Feature-Driven Development, Pragmatic Programming,
  - and others sympathetic to the need for an alternative to documentation driven, heavyweight software development processes”

Set of values and principles shared by the agile community.

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

The Agile Manifesto

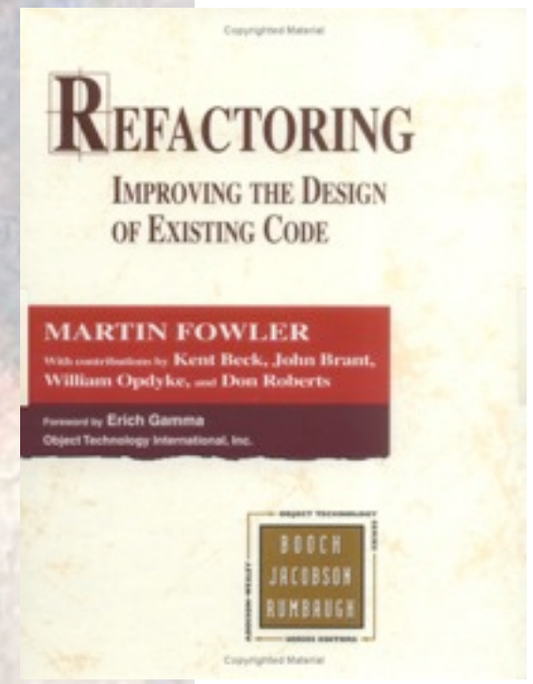
Manifesto for Agile Software Development

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.

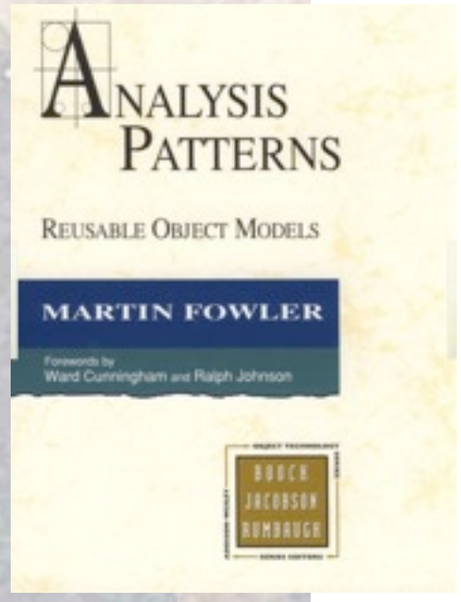
Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas



JUnit

JUnit

WikiWikiWeb



The Agile Manifesto

- **“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.”

The Agile Manifesto - 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.

The Agile Manifesto - Principles

- **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**.

The Agile Manifesto - Principles

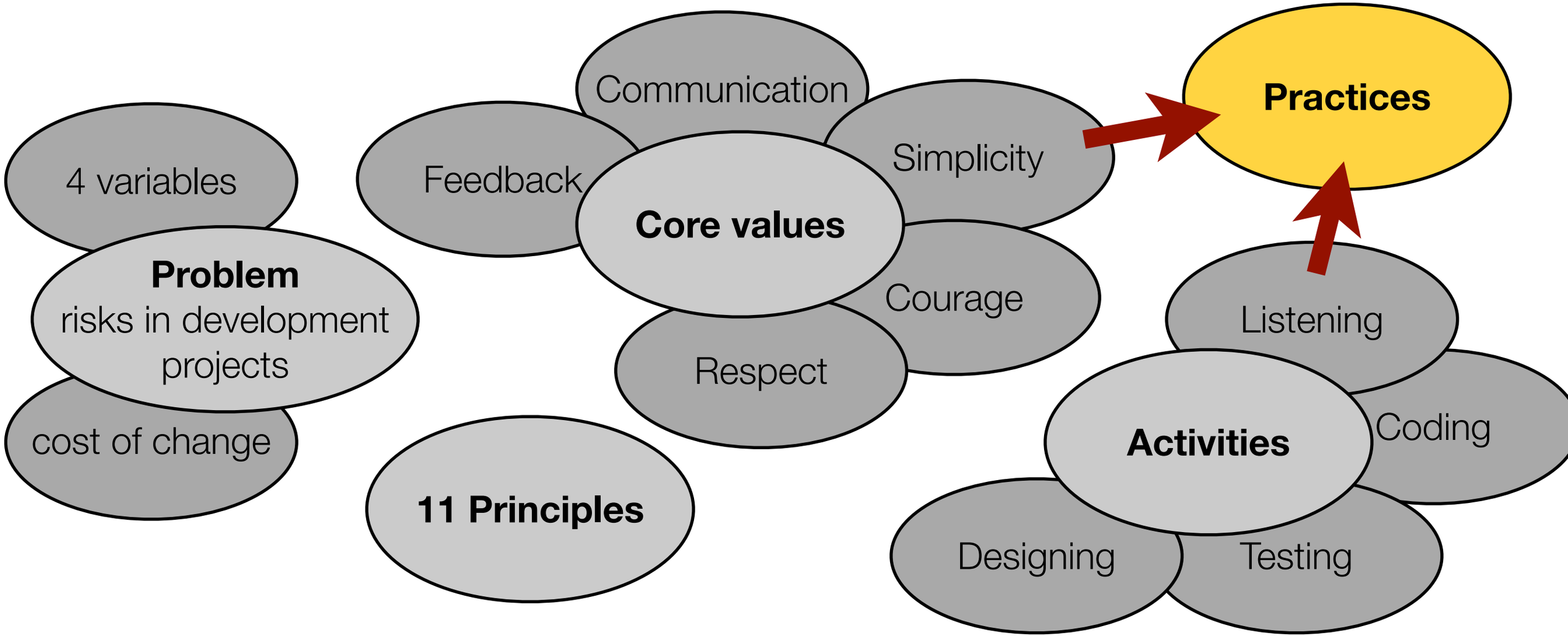
- Working software is the primary **measure of progress**.
- Agile processes promote **sustainable development**. The sponsors, developers, and users should be able to **maintain a constant pace indefinitely**.

The Agile Manifesto - Principles

- **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.

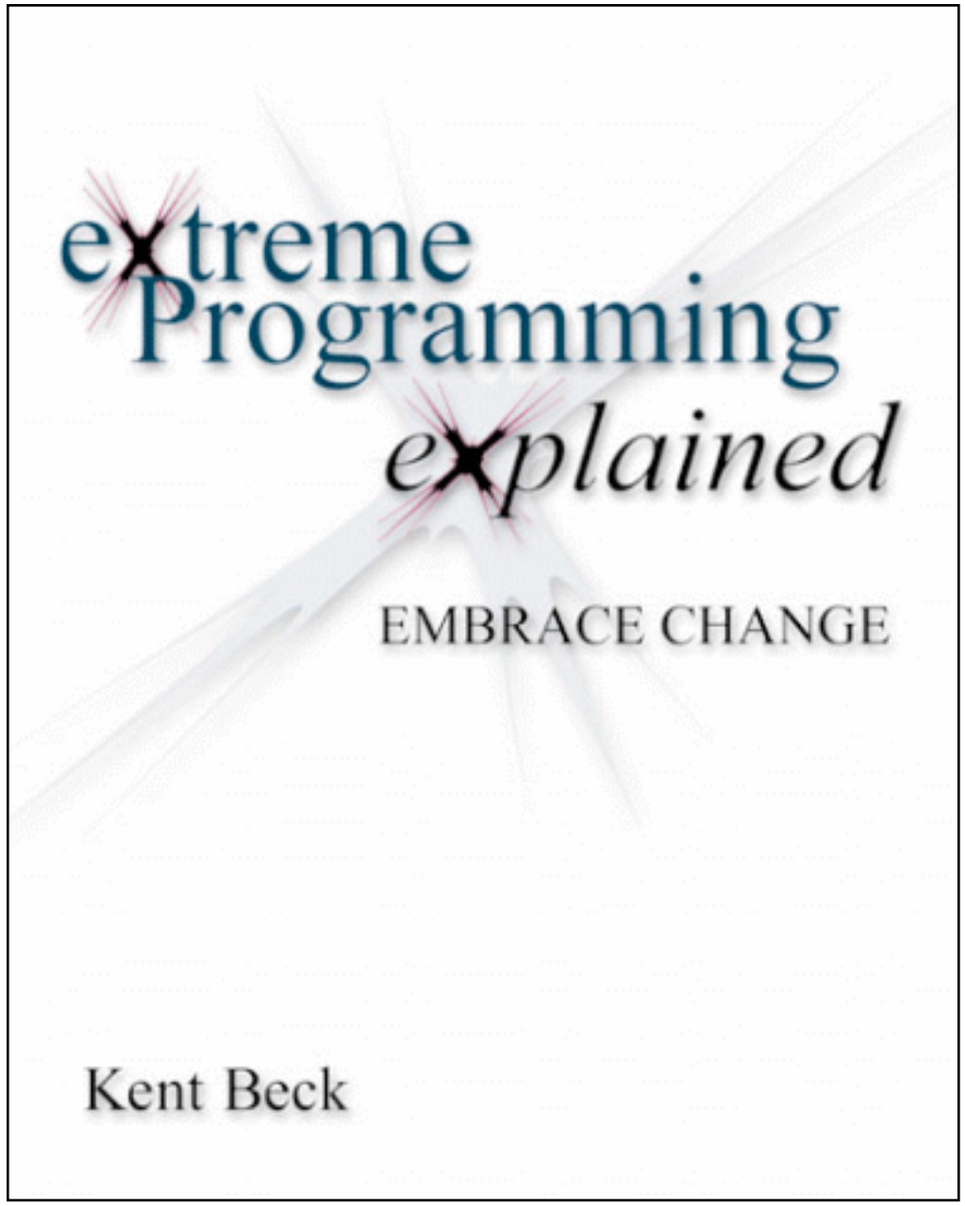
eXtreme Programming (XP)

- Kent Beck
- Chrysler Comprehensive Compensation System (C3), 1996
- “eXtreme Programming explained - embrace change”, published in 1999, 2nd edition in 2004



The XP reference book

- Most of the content for these slides comes from the first edition of the eXtreme Programming reference book.



The Problem: risks in development projects

- Schedule slips
- Project canceled
- System does not evolve gracefully (defect rate increases)
- Defect rate
- Business misunderstood
- Business changes
- False feature rich
- Staff turnover

The role of XP is to give us principles and practices in order to deal with these risks!

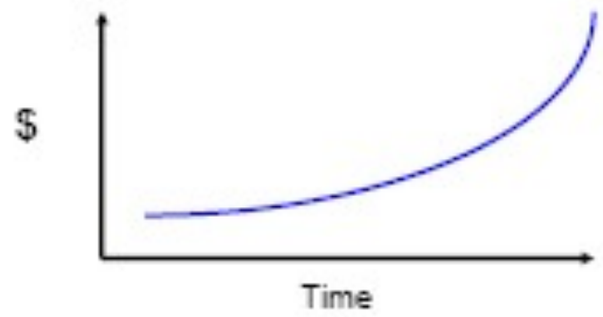
Developing software: 4 variables

- **4 variables:**
  - Time
  - Money
  - Quality
  - Scope
- External forces (customers, managers) can set the values for 3 of the variables.
- The development team sets the value for the 4th one.

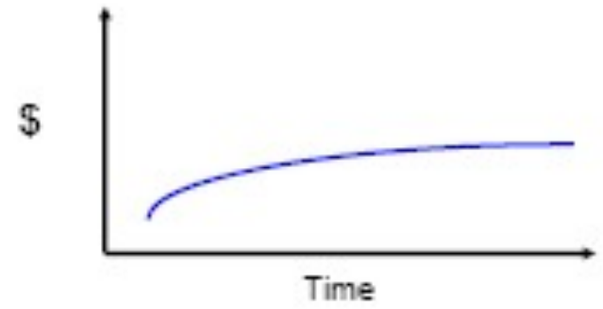
Which variable do you use as a control variable?

Cost of change

Traditional view



Agile view



The Values of XP: simplicity

- We will do what is needed and asked for, but no more.
- **This will maximize the value created for the investment made to date.**
- We will take small simple steps to our goal and mitigate failures as they happen.
- We will create something we are proud of and maintain it long term for **reasonable costs**.

http://www.extremeprogramming.org/values.html

The Values of XP: communication

- Everyone is **part of the team** and we **communicate face to face daily**.
- We will **work together on everything** from requirements to code.
- We will create the best solution to **our problem** that we can together.

http://www.extremeprogramming.org/values.html

The Values of XP: feedback

- We will take every iteration **commitment** seriously by delivering **working software**.
- We **demonstrate our software early and often** then **listen carefully** and make any changes needed.
- We will talk about the project and **adapt our process** to it, not the other way around.

http://www.extremeprogramming.org/values.html

The Values of XP: respect

- Everyone gives and feels the respect they deserve as a **valued team member**.
- **Everyone contributes value even if it's simply enthusiasm**.
- Developers **respect the expertise of the customers** and vice versa.
- Management respects our right to **accept responsibility and receive authority** over our own work.

http://www.extremeprogramming.org/values.html

The Values of XP: courage

- We will **tell the truth about progress** and estimates.
- We **don't document excuses** for failure because we plan to succeed.
- We don't fear anything because **no one ever works alone**.
- We will **adapt to changes** when ever they happen.

http://www.extremeprogramming.org/values.html

XP: basic principles

- **Fundamental principles**
  - Rapid feedback
  - Assume simplicity
  - Incremental change
  - Embracing change
  - Quality work
- **Other principles**
  - Teach learning
  - Small initial investment
  - Play to win
  - Concrete experiments
  - Open, honest communication
  - Work with people’s instincts, not against them
  - Accepted responsibility
  - Local adaptation
  - Travel light
  - Honest measurement

Learning to drive

- “We need to control the development of software by making many small adjustments, not by making a few large adjustments, kind of like driving a car.
- This means that we will need the feedback to know when we are a little off, we will need many opportunities to make corrections, and we will have to be able to make those corrections at a reasonable cost”.

XP Practices (v1)

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40 hours week
- On-site customer
- Coding standards

XP Practices (v1)

- **The Planning Game**
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40 hours week
- On-site customer
- Coding standards

- Balance between **business** and **technical** considerations
- **Business** people decide about:
  - Scope + Priority + Composition of releases + Dates of releases
- **Technical** people decide about:
  - Estimates + Consequences + Process + Detailed Scheduling

XP Practices (v1)

- The Planning Game
- **Small releases**
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40 hours week
- On-site customer
- Coding standards

- Every release should be as small as possible, containing the **most valuable business requirements**.
- The release has to make sense as a whole (no half-working features).
- Better to release once a month than twice a year.

XP Practices (v1)

- The Planning Game
- Small releases
- **Metaphor**
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40 hours week
- On-site customer
- Coding standards

- Everybody on the team needs to have a “**common understanding**” for the system.
- Everybody on the team needs to have a “**shared vocabulary**”.
- This applies to **technical** and **non-technical** people.
- What are the basic elements of the system and what are their relationships?

XP Practices (v1)

- The Planning Game
- Small releases
- Metaphor
- **Simple design**
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40 hours week
- On-site customer
- Coding standards

- The **right design** for a software system is one that:
  - Runs all tests.
  - Has no duplicated logic.
  - Has the fewest possible classes and methods.
- “Put in what need when you need it”.
- **Emergent**, **growing design**; no big design upfront (through refactoring)

XP Practices (v1)

- The Planning Game
- Small releases
- Metaphor
- Simple design
- **Testing**
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40 hours week
- On-site customer
- Coding standards

- Any program feature without an **automated test** simply doesn't exist.
- The tests become part of the system.
- The tests allow the system to **accept change**.
- **Development cycle**:
  - Listen (requirements)
  - Test (write first)
  - Code (simplest)
  - Design (refactor)

XP Practices (v1)

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- **Refactoring**
- Pair programming
- Collective ownership
- Continuous integration
- 40 hours week
- On-site customer
- Coding standards

- When implementing a feature, ask yourself if there is a way to change the existing source code, so that implementing the feature is easier.
- Automated tests enable programmers to refactor without fear.

XP Practices (v1)

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- **Pair programming**
- Collective ownership
- Continuous integration
- 40 hours week
- On-site customer
- Coding standards

- All production code is written by **two people** looking at **one screen**, with one keyboard and one mouse.
- **Two roles.** The programmer on the keyboard focuses on the current method. The other programmer thinks about the broader context (refactoring, etc.)
- **Pairs change frequently**.

XP Practices (v1)

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- **Collective ownership**
- Continuous integration
- 40 hours week
- On-site customer
- Coding standards

- **Anybody** who sees an **opportunity** to **add value** to **any portion** of the code is required to do so at any time. TTL.
- Everybody **takes responsibility** for the whole of the system. Not everyone knows every part equally well, but everyone knows something about every part.

XP Practices (v1)

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- **Continuous integration**
- 40 hours week
- On-site customer
- Coding standards

- Code is integrated and tested **at least once a day** (sometimes more).
- **Build process must me automated**, on a dedicated machine.
- Automated **tests** are run and make it possible to identify problems early.

XP Practices (v1)

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- **40 hours week**
- On-site customer
- Coding standards

- **Sustainable** development. Effort should be **spread out** evenly.
- Extended periods of overtime have a negative impact on productivity.
- Goal: be **fresh** every morning, be tired and **satisfied** every evening.
- Not being in front of a computer does not mean forgetting about the system... taking a step back often leads to “Aha!” moments.

XP Practices (v1)

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40 hours week
- **On-site customer**
- Coding standards

- A real customer must be physically with the team, available to answer their questions.
- Real customer = user who will use the system.
- The real customer does not work on the project 100% of his time, but needs to be “there” to answer questions rapidly.
- The real customer also help with prioritization.

XP Practices (v1)

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40 hours week
- On-site customer
- **Coding standards**

- **Collective ownership** + **constant refactoring** means that coding practices must be unified in the team.

XP Second Edition: 13 primary practices

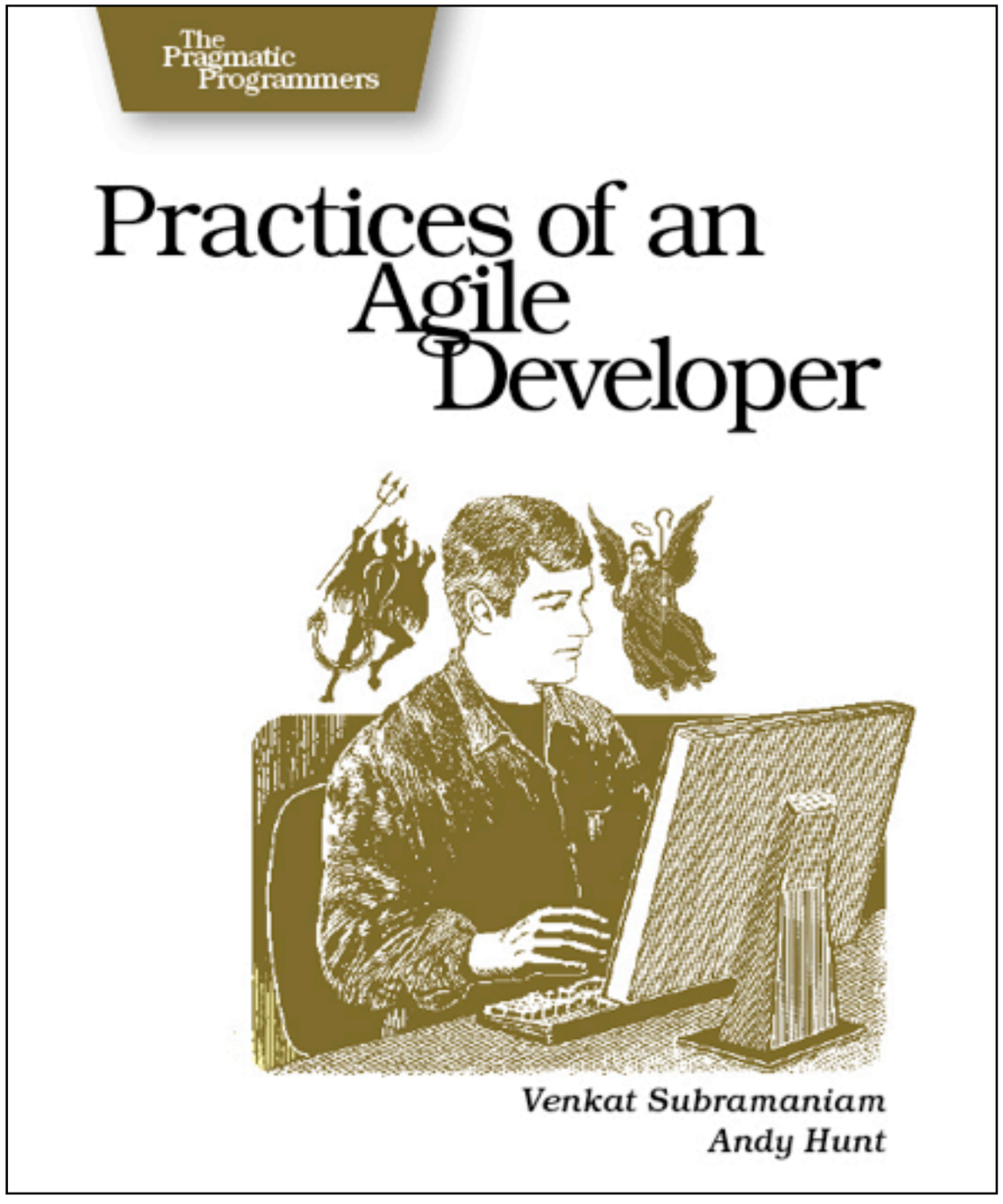
- Sit Together
- Whole Team
- **Informative Workspace**
- Energized Work
- Pair Programming
- Stories
- **Weekly Cycle**
- **Quarterly Cycle**
- **Slack**
- Ten-Minute Build
- Continuous Integration
- Test-First Programming
- Incremental Design

XP Second Edition: 11 corollary practices

- Real Customer Involvement
- Incremental Deployment
- Team Continuity
- Shrinking Teams
- Root-Cause Analysis
- Shared Code
- Code and Tests
- Single Code Base
- Daily Deployment
- Negotiated Scope Contract
- Pay-Per-Use

Practices of an Agile Developer

- This is a collection of agile work practices, with examples.
- Beginning agility (4)
- Feeding agility (5)
- Delivering what users want (9)
- Agile feedback (6)
- Agile coding (8)
- Agile debugging (5)
- Agile collaboration (8)
- Epilogue: moving to agility (5)



Conclusion: no silver bullet

- There is **no “magic” process** that would work exactly the same way for every project, in every environment.
- Agile methodologies and XP describe **core values and key principles** that you need to integrate and **customize** in your particular **context**.
- Agile teams need to **continuously reflect** on their work.
- XP looks like it is **less “formal”** than traditional methodologies. But while there are certainly less roles, less workflows and less artifacts, XP requires **a lot of discipline** to work well.

© Scott Adams, Inc./Dist. by UFS, Inc.