SSW-555: Agile Methods for Software Development

Introduction

Extreme Programming/XP

Dr. Richard Ens
Software Engineering
School of Systems and Enterprises

Just for fun…

Today's topics

Software development method comparison

Motivations for agile methods

- Rational Unified Process (RUP)
- Boehm's risk exposure comparison

Overview of Extreme Programming (XP)

Acknowledgements

Lecture material comes from a variety of sources, including:

https://www.tutorialspoint.com/sdlc/sdlc_quick_guide.htm

Software Engineering, 10th Edition, Ian Sommerville

Scott W. Ambler www.ambysoft.com/surveys/

"Get ready for agile methods, with care" by Barry Boehm, *IEEE Computer*, January 2002.

http://agilemanifesto.org/

http://www.extremeprogramming.org/

How much planning?

What's the right level of planning for software projects?

- It depends on the task!

How should we decide?

- What's the domain?
- How complete are the requirements?
- How stable are the requirements?
- What's the cost of doing the wrong thing?
- What's the cost of doing the right thing too slowly?
- What are the risks?
- What are the rewards?

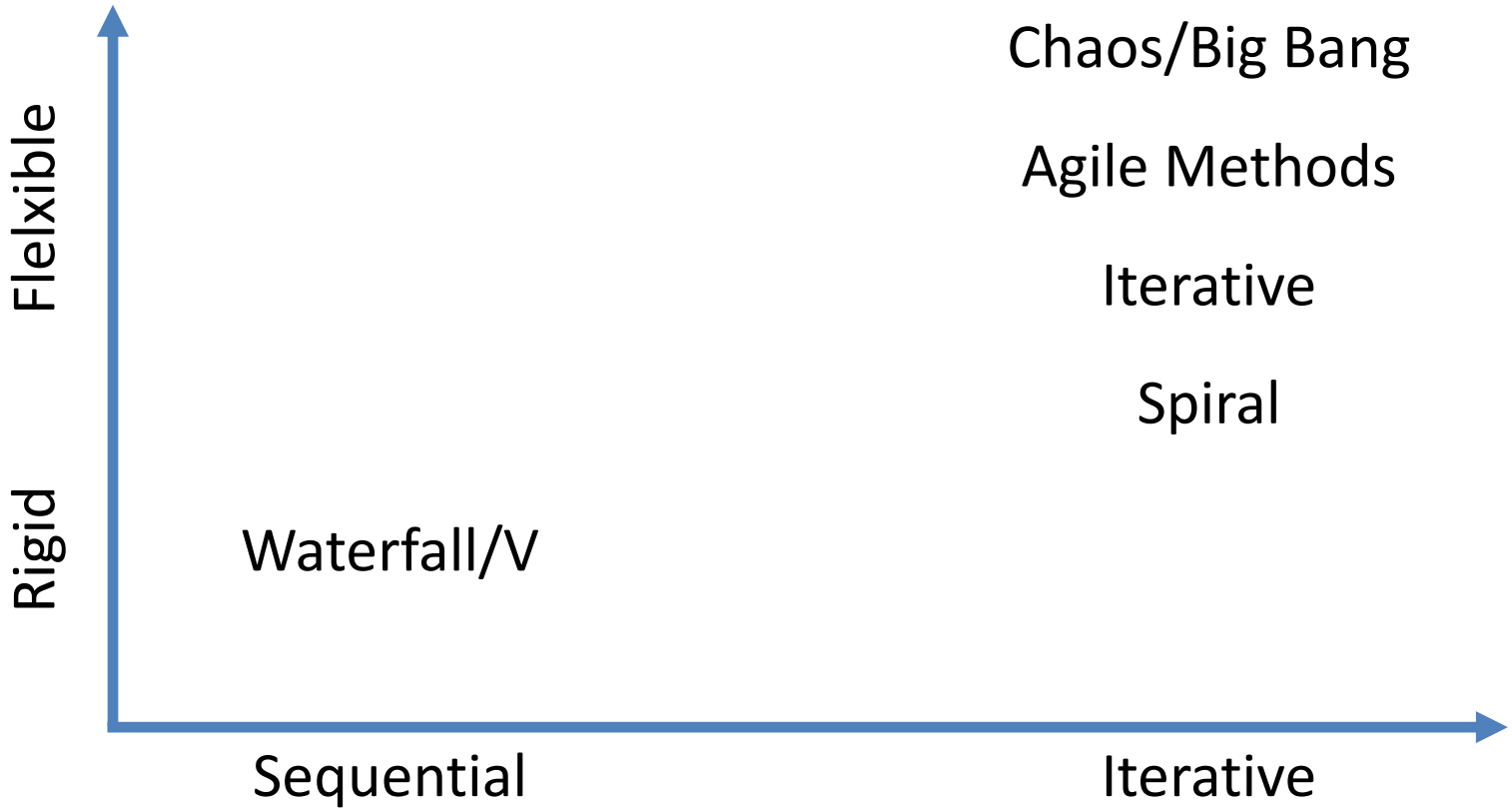
Software Development Life Cycle (SDLC)

What problems are we trying to solve in any SDLC?

Software specification	• What functionality must we support?
Software development	• How do we create the software that delivers the functionality?
Software validation	• How do we verify that the software does what it's supposed to do?
Software evolution	• How does the software evolve to meet customer needs?

Source: Software Engineering, 10th Edition, Ian Sommerville

SDLC Methods



Waterfall Model

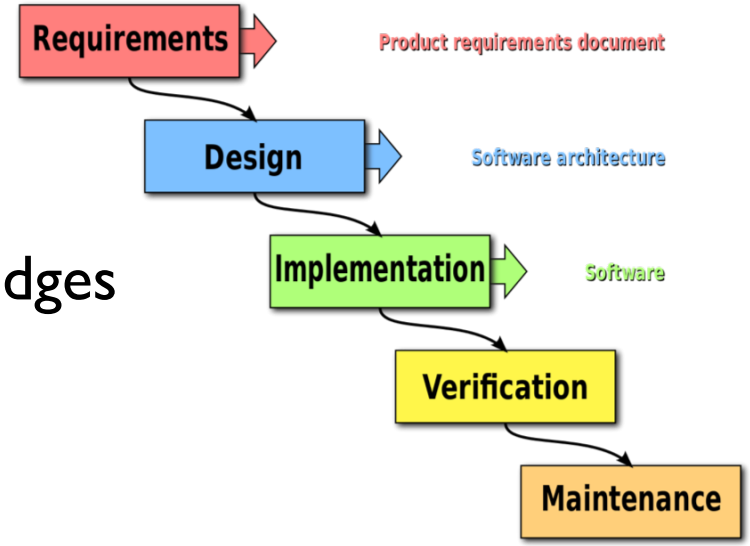
Waterfall Model

- Plan everything in sufficient detail so we can get it right the first time
  - <sarcasm> not likely </sarcasm>

Popular approach with traditional engineering problems, e.g. building bridges

Very formal process

- Extensive documentation
- Serial execution
  - Complete each stage before starting next
  - Very difficult to go back to earlier stages
- Strict handoffs between stages
  - "Signed off in blood"

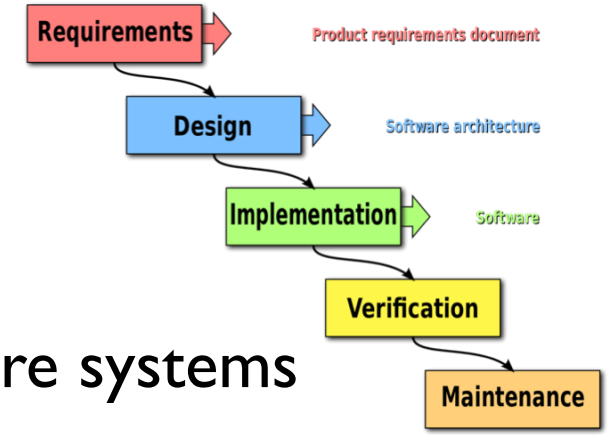


https://upload.wikimedia.org/wikipedia/commons/thumb/e/e2/Waterfall_model.svg/1280px-Waterfall_model.svg.png

Waterfall Model

When is Waterfall appropriate?

- When we must get it right the first time!
- Cost of failure is very high
- Complete requirements



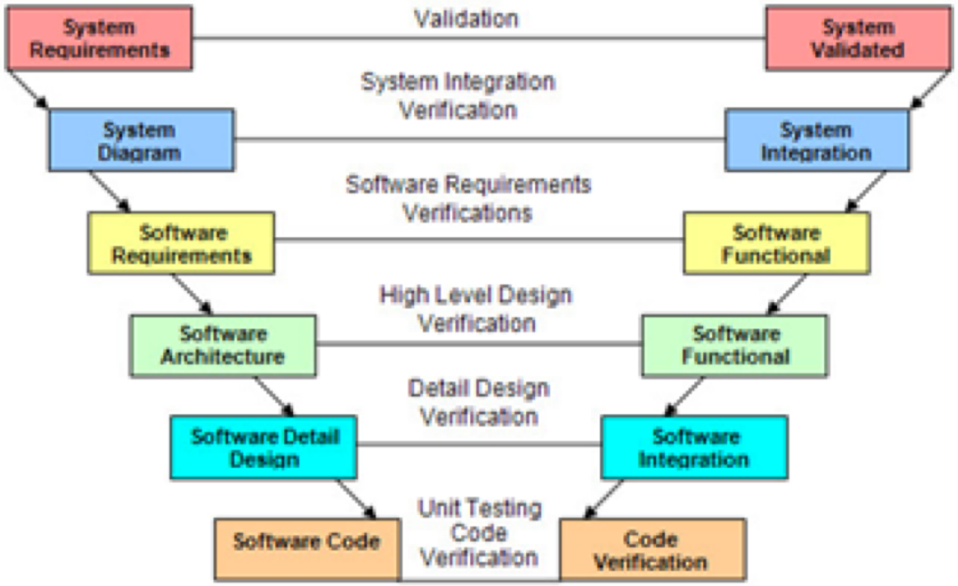
Examples:

- Interacting with embedded hardware systems
  - Very difficult/expensive to change the hardware components
- Critical systems with extreme safety or security requirements
  - e.g. airplanes, self-driving cars, …
- Very large, multi-organizational software projects

V (Verification/Validation) SDLC Model

Add verification/testing to each step of the Waterfall Model

Must complete rigorous testing before proceeding to next step



https://sites.google.com/site/advancedsofteng/software-acquisition/software-development-lifecycle-approaches

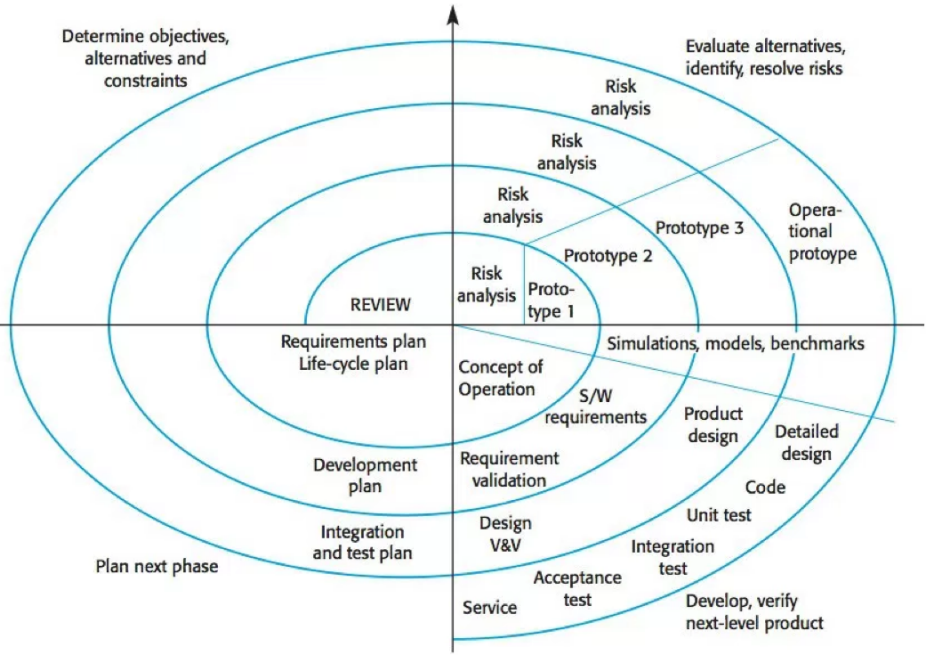
Boehm's Spiral Model

Recognizes limitations of Waterfall
- Hard to get it right the first time…
- Customers don't know what they want

Adds focus on risk assessment

Encourages incremental development and iterations
- Learn from previous iterations

Each spiral includes:
- Objective setting
- Risk assessment and reduction
- Development and validation
- Planning



http://iansommerville.com/software-engineering-book/web/spiral-model/

Iterative models SDLC

Recognizes limitations of Waterfall
- Hard to get it right the first time…
- Customers don't know what they want

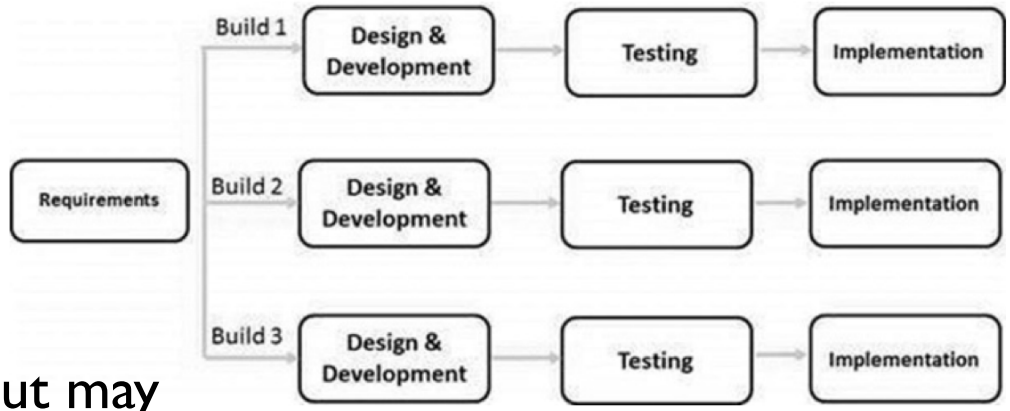
Identify requirements up front

Build subsets of requirements
- Sequentially or in parallel with multiple groups of developers

Requirements are mostly stable but may change

Works well for large projects that can be developed in parallel efforts

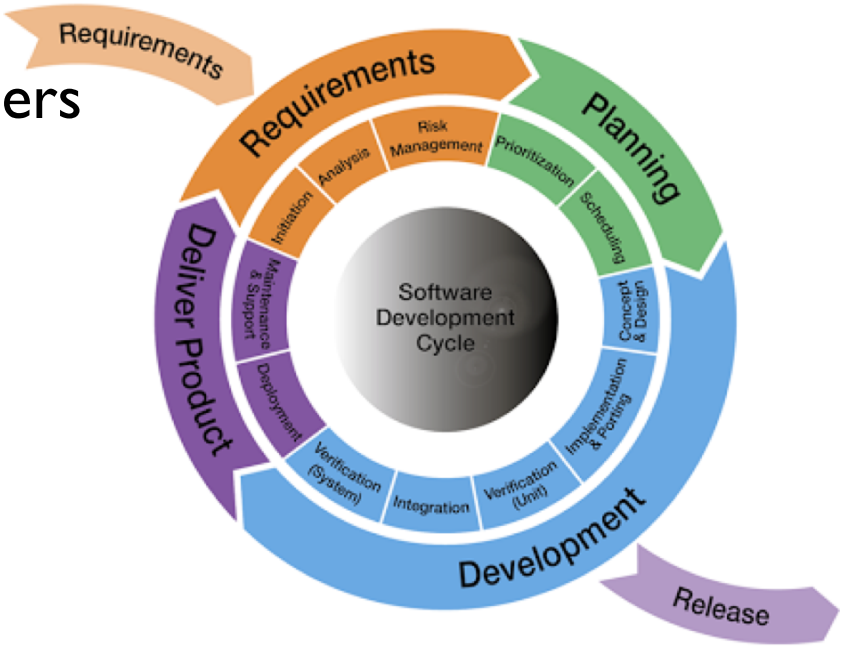
Delivers early functionality to customers for review



https://www.tutorialspoint.com/sdlc/sdlc_quick_guide.htm

Agile Methods SDLC

- Frequent iterations and deliverables
- Close collaboration between customers and developers
  - Customer is a critical partner in the process rather than an observer
- Supports changing requirements
- Frequent reflection and continuous improvement
  - What are we doing well?
  - What can we improve?
  - Learn and improve from experience



https://www.pinterest.com/pin/553168766708998354/

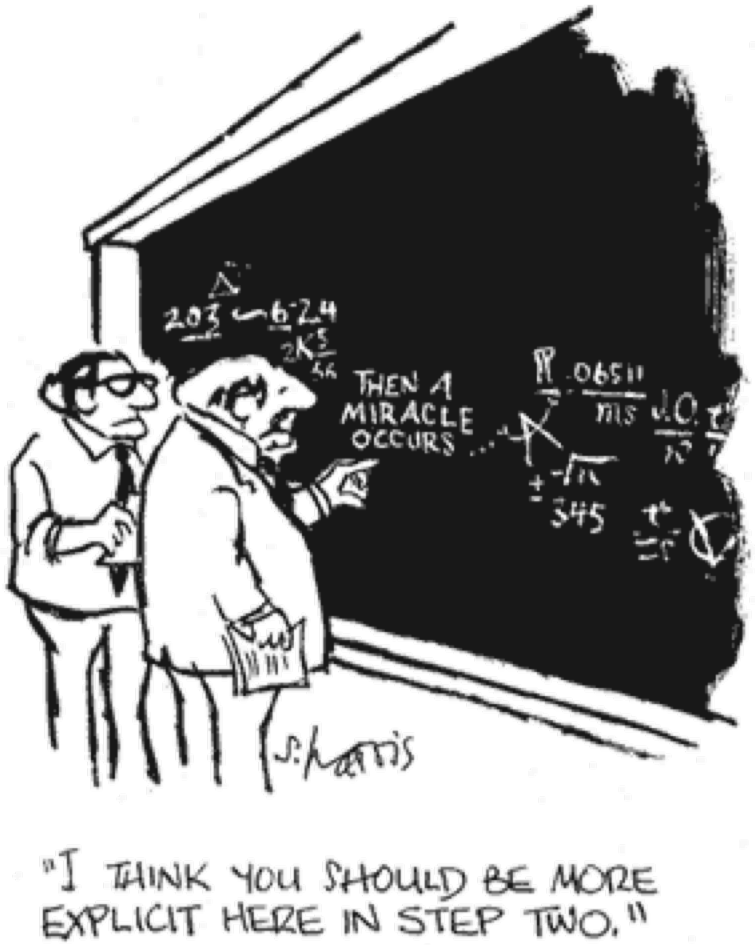
Big Bang/Chaos SDLC

Little to no planning

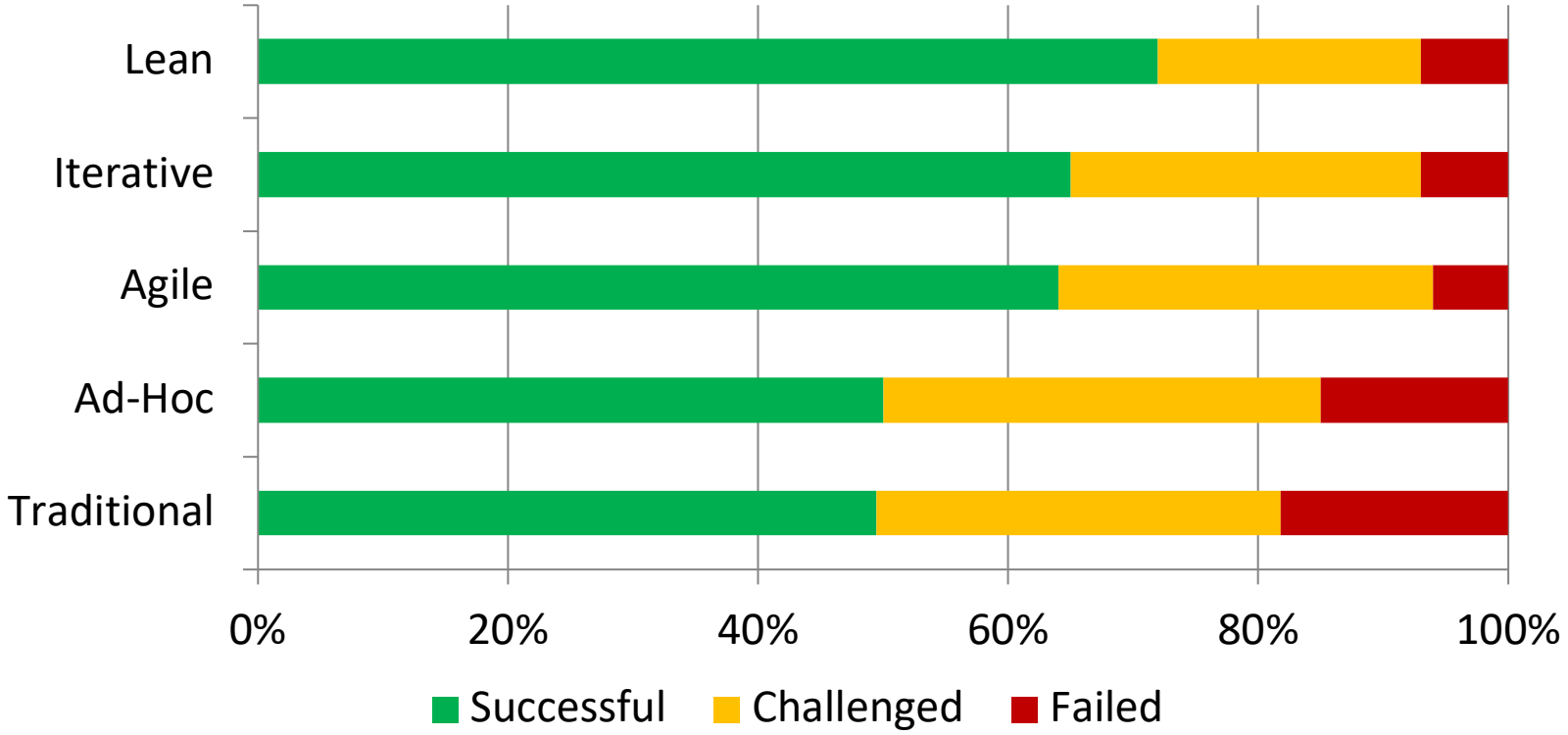
Figure it out as you go

Typically used for very small projects

Not highly recommended…

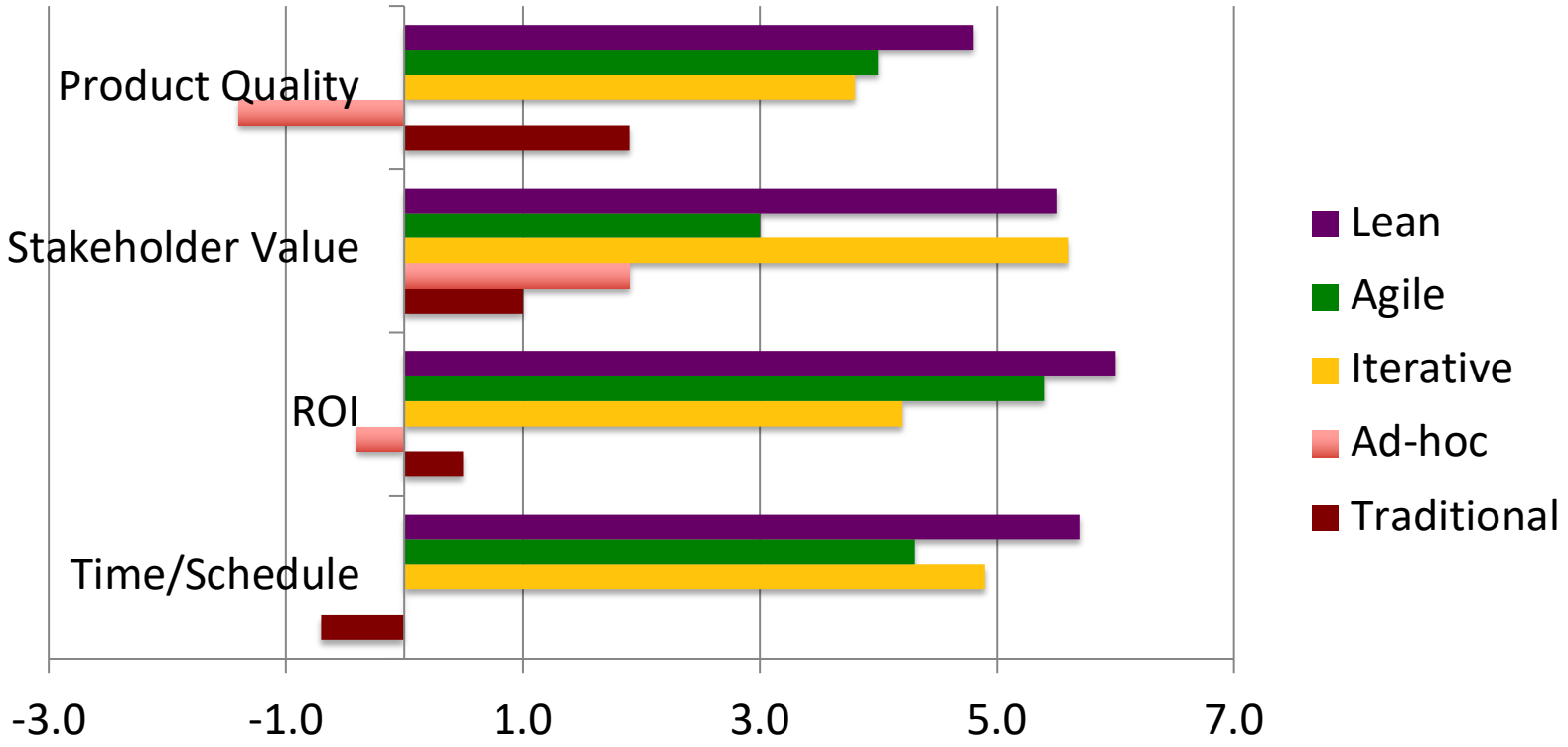


Comparing Software Development Paradigms: 2013



SCOTT AMBLER + Associates

Comparing Delivery Paradigms



SCOTT AMBLER + Associates

Software development in the 1990s

Large projects (> 40 people, > 1 year)

- Used plan-based methods, such as RUP (Rational Unified Process)
- More incremental than waterfall, but heavy process
- Focus on quality, discipline, process improvement

Medium-sized projects (7-40 people, 3-12 months)

- Some used plan-based methods, some used ad hoc methods
- Struggled to reduce process overhead while maintaining quality

Small projects (< 7 people, < 3 months)

- Many used ad hoc methods (or no methods)
- Struggled to achieve greater speed

Software development before Agile

1960s:

- Original software crisis leads to "software engineering"
- Apply techniques from other engineering disciplines to software

1970s:

- Adoption of traditional engineering methods, such as the waterfall model

1980s:

- Many attacks on complexity (OO, CASE tools, formal methods, iterative process models, process maturity)
- "No silver bullet" published by Brooks
- Software market expands from DoD and large business to home PCs

1990s:

- WWW, dot-com boom, and Internet time
- Backlash against heavy process

Rational Unified Process (RUP)

Developed at Rational Software in late 1990s after acquisition of several Object Oriented companies

Based on 6 ***Best Practices*** of Software Engineering

- Develop iteratively
- Manage requirements
- Use component-based architectures
- Model software visually (UML)
- Continuously verify software quality
- Control changes

RUP Best Practices

Develop software iteratively

- Solutions are too complex to get right in one pass
- Use an iterative approach and focus on the highest ***risk*** items in each pass
- Customer involvement
- Accommodate changes in requirements

Manage requirements

- Use cases and scenarios help to identify requirements
- Requirements provide traceable thread from customer needs through development to end product

RUP Best Practices

Use component based architectures

- Creating and base lining an architecture is a gate for development
- Architecture should be flexible to accommodate change
- Focus on reusable, component-based software

Visually model software

- Capture structure and behavior in Unified Modeling Language (UML)
- UML helps to visualize the system and interactions

RUP Best Practices

Verify software quality

Verification and Validation is part of the process, not an afterthought

Focus on reliability, functionality, and performance

Control changes to software

Change is inevitable

Actively manage the change request process

Control, track, and monitor changes

RUP project lifecycle phases

Inception: scope system for cost and budget, create basic use case model

Elaboration: mitigate risks by elaboration of use case model and design of software architecture

Construction: implement and test software

Transition: plan and execute delivery of system to customer

Each phase ends with a milestone when stakeholders review progress and make go/no-go decisions

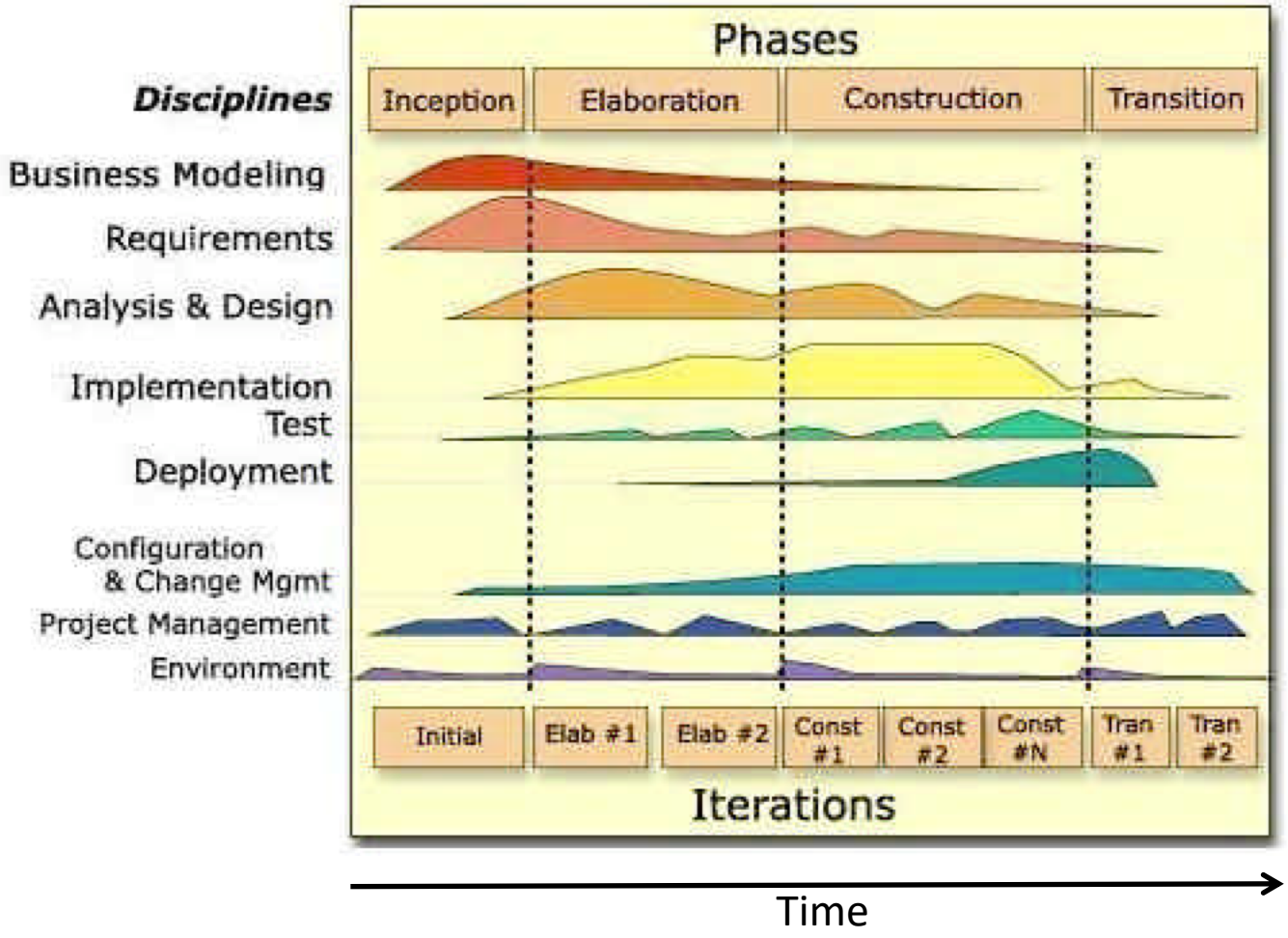
RUP supporting disciplines

Configuration and Change Management: manage access to project work products

Project Management: manage risks, direct people, coordinate with other stakeholders

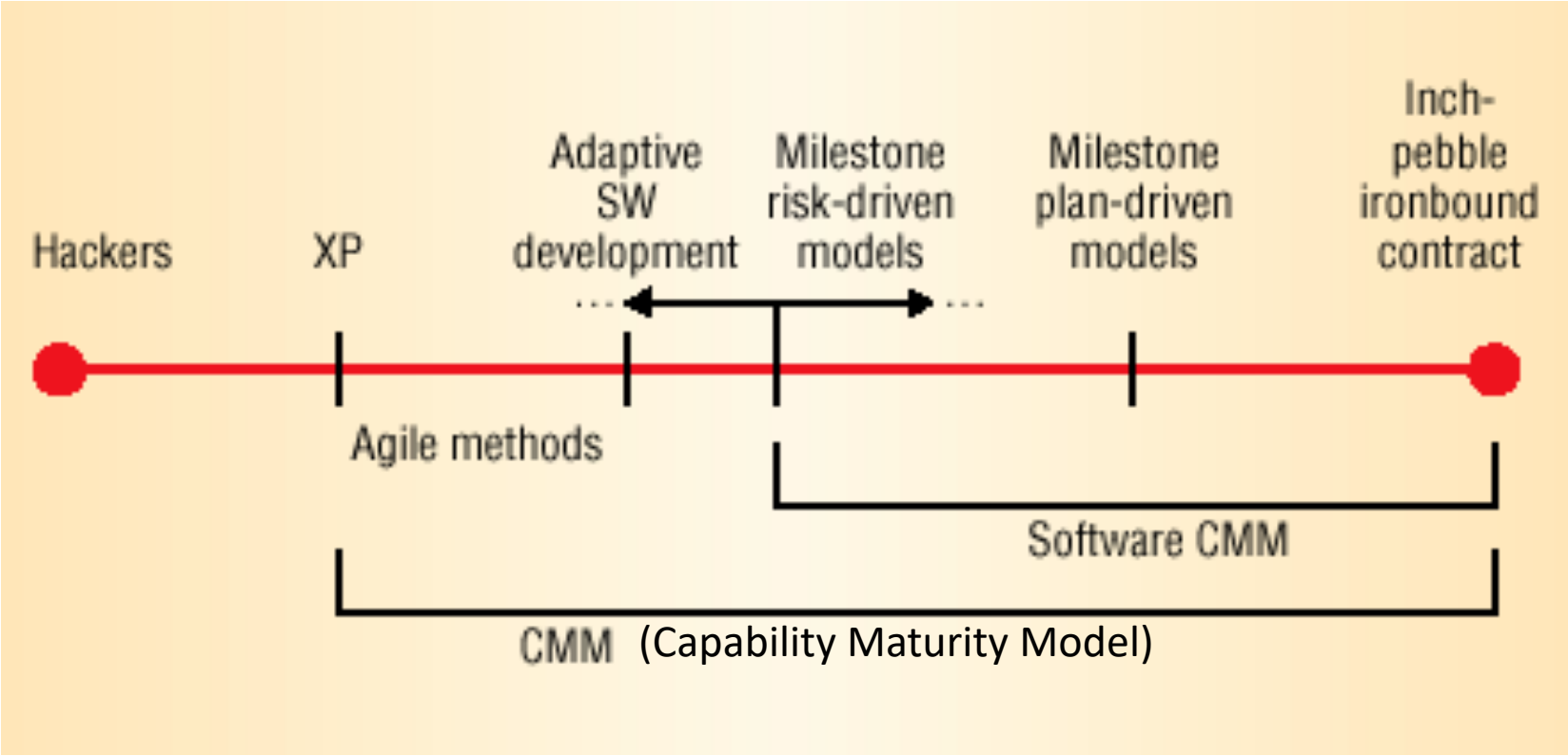
Environment: ensure that process, guidance and tools are available

RUP phases, iterations and disciplines



RUP®
Rational Unified Process®

Spectrum of methods to meet different project needs



Source: "Get ready for agile methods, with care" by Barry Boehm, *IEEE Computer*, January 2002.

Boehm's risk exposure profile:
How much planning is enough?

Need to get to market quickly

RE: Risk Exposure
P(L): Probability of Loss
S(L): Size of Loss

Black curve:
Inadequate plans

Red curve:
Loss of market share

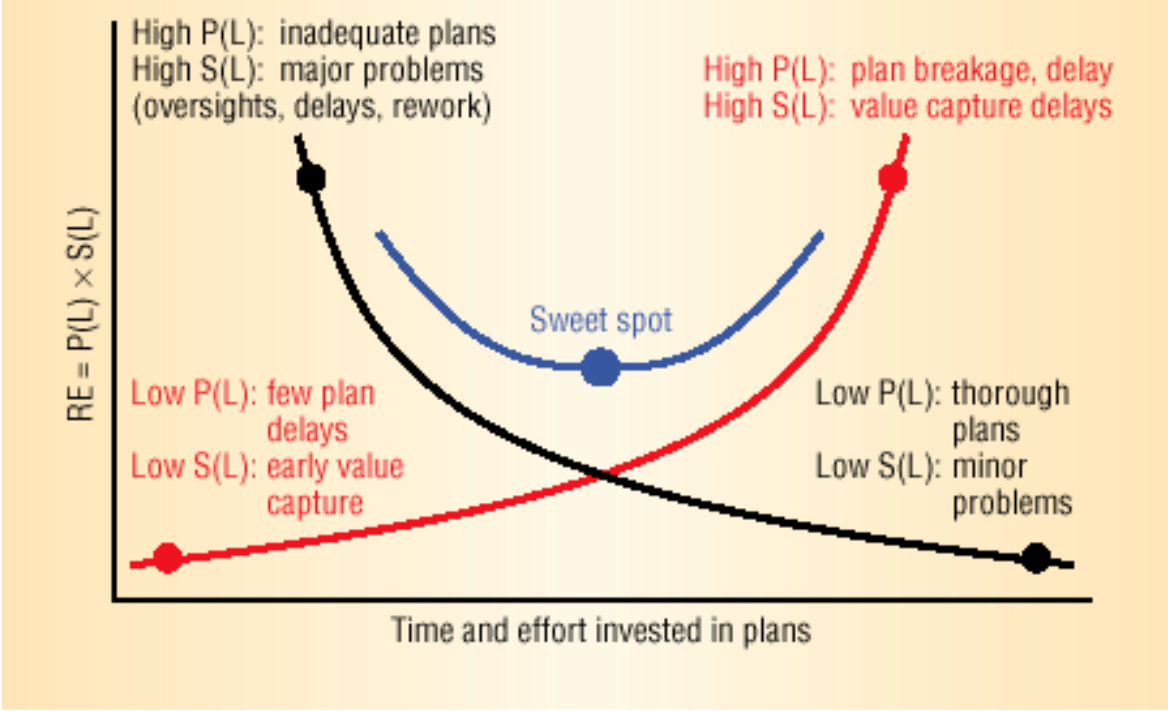


Figure 2. Risk exposure (RE) profile. This planning detail for a sample e-services company shows the probability of loss P(L) and size of loss S(L) for several significant factors.

Source: "Get ready for agile methods, with care" by Barry Boehm, *IEEE Computer*, January 2002.

Safety-critical profile

RE: Risk Exposure
P(L): Probability of Loss
S(L): Size of Loss

Black curve:
Inadequate plans

Red curve:
Loss of market share

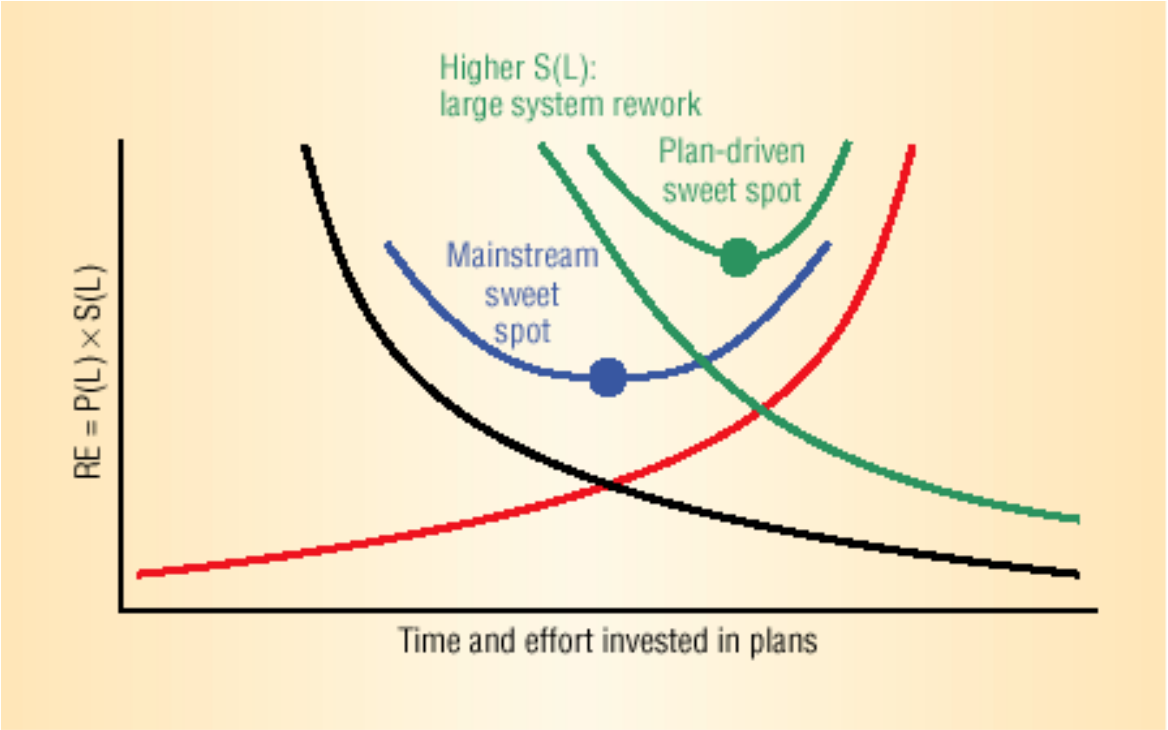


Figure 4. Comparative RE profile for a plan-driven home-ground company that produces large, safety-critical systems.

Source: "Get ready for agile methods, with care" by Barry Boehm, *IEEE Computer*, January 2002.

Agile profile

RE: Risk Exposure
P(L): Probability of Loss
S(L): Size of Loss

Black curve:
Inadequate plans

Red curve:
Loss of market share

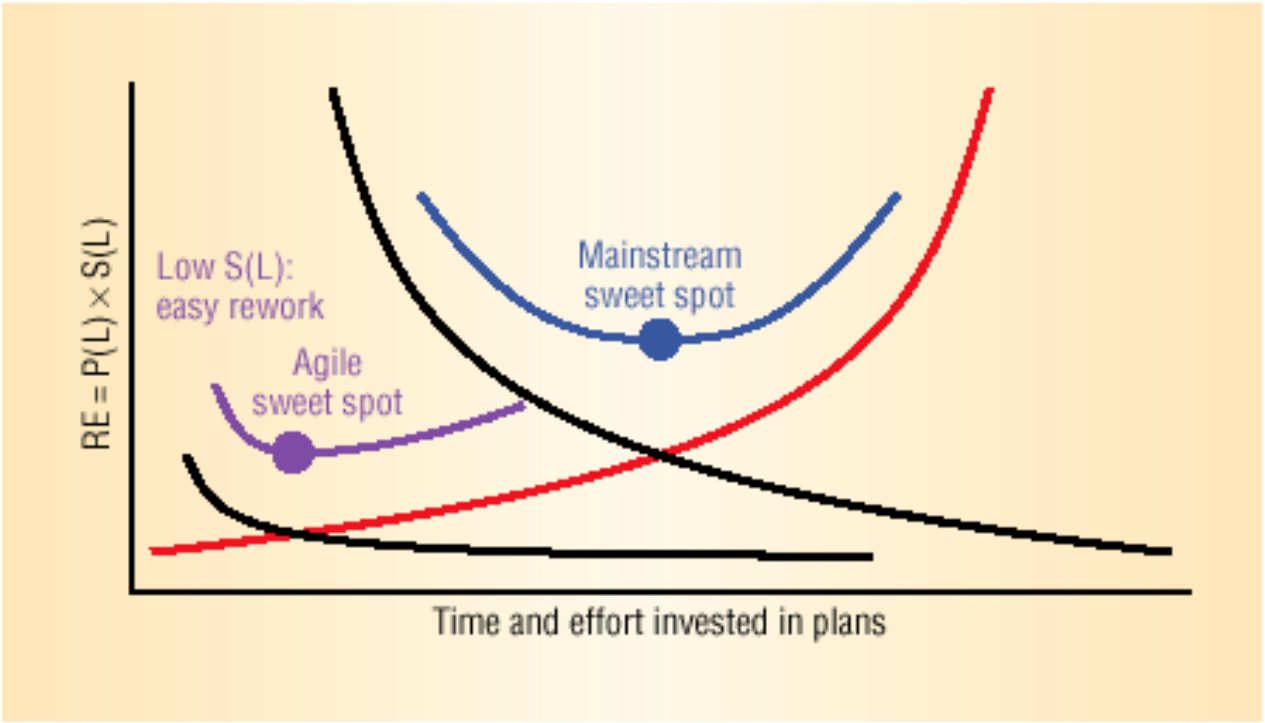


Figure 3. Comparative RE profile for an agile home-ground company with a small installed base and less need for high assurance.

Source: "Get ready for agile methods, with care" by Barry Boehm, *IEEE Computer*, January 2002.

Agile Manifesto (2001)

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.

- 12 Principles behind the Agile Manifesto
- http://agilemanifesto.org/

12 Principles of the Agile Manifesto

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.

http://agilemanifesto.org/principles.html

12 Principles of the Agile Manifesto

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.

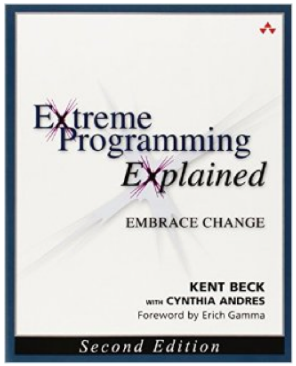
Simplicity--the art of maximizing the amount of work not done--is essential.

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.

http://agilemanifesto.org/principles.html

Extreme Programming (XP)



Created by Kent Beck while working on a project for Chrysler in the late 1990s

Collaborators: Ward Cunningham and Ron Jeffries

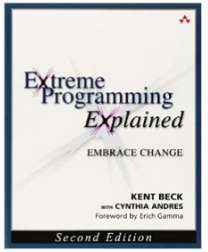
One of the most well-known agile methods

Takes best practices to extreme levels

"Hard to get an XP t-shirt" - Rowland

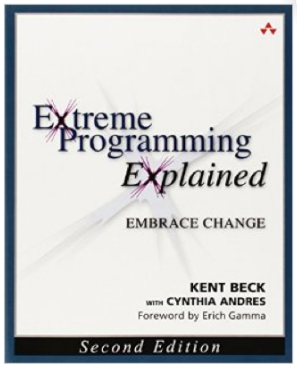
http://www.extremeprogramming.org/

12 Extreme Programming Practices



- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- Sustainable pace
- Whole team
- Coding standards

The Planning Game



Business people decide:

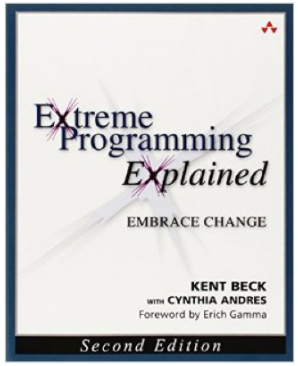
- scope
- priority
- release dates

Technical people decide:

- estimates of effort
- technical consequences
- process
- detailed scheduling

Requires a close collaboration between customers and developers not found in traditional plan driven approaches

Small releases



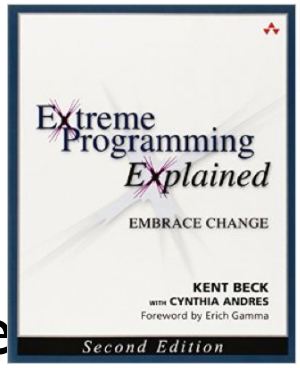
Every release should be as small as possible

Every release must completely implement its new features

Every release should contain the ***most valuable business* features**

- Contrast with RUP where you focus on the biggest risk first

Metaphor

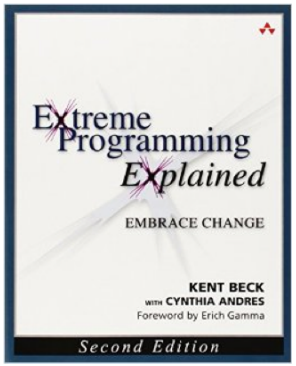


Metaphor is a simple explanation of the proje

- Agreed upon by all members of the team
- Simple enough for customers to understand
- Detailed enough to drive the architecture

Metaphor replaces architecture as 10,000 feet view of the system

Simple design



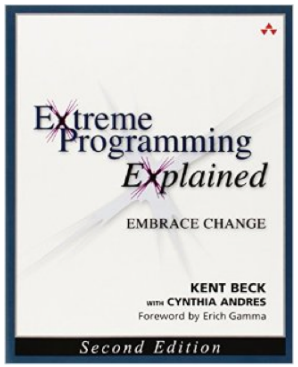
Successfully runs all the tests

Has no duplicated logic

States every intention important to programmers

Has the fewest possible classes and methods

Testing

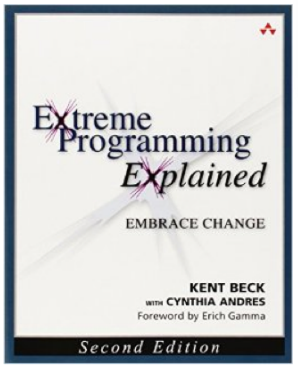


Any feature without an automated test does not exist

Programmers need confidence in correct operation

Customers need confidence in correct operation

Refactoring



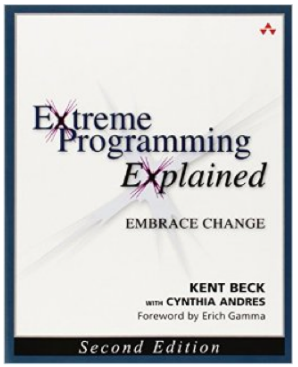
Rewrite or restructure the code to improve the implementation

Always ask if there is a way to make the program simpler

When the system requires duplication of code, it is asking for refactoring

Can always find a series of small, low-risk steps

Pair programming



All code written with two people at one machine

Driver:

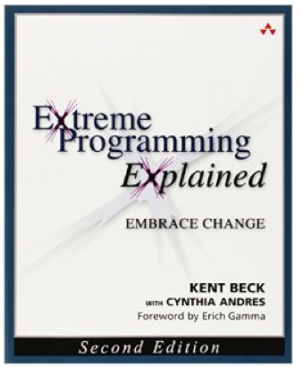
- thinks about best way to implement

Navigator:

- thinks about viability of whole approach
- thinks of new tests
- thinks of simpler ways

Switch roles frequently

Collective ownership

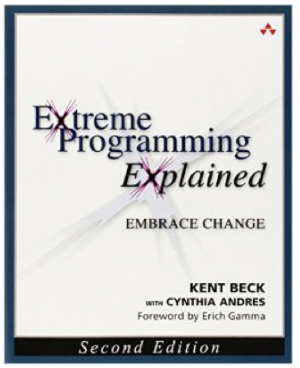


Anybody who sees an opportunity to add value to any portion of the code is required to do so

Everyone knows something about everything

Everyone feels obligated to make improvements

Continuous integration



Integrate and test every few hours, at least once per day

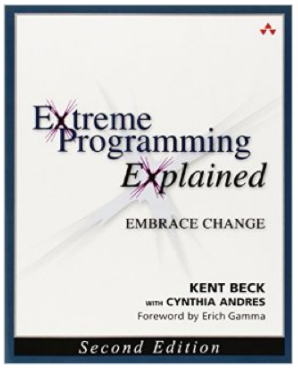
- Don't wait until the very end to begin integration

All tests must pass

Easy to tell who broke the code

- Problem is likely to be in code that was most recently changed

Sustainable pace (40-hour week)

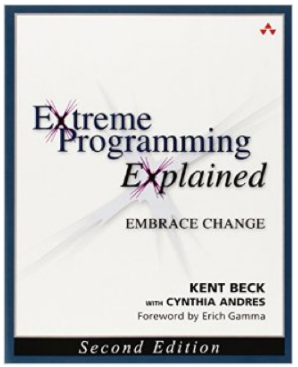


People should be fresh and eager every morning

Overtime is a symptom of a serious problem

XP only allows one week of overtime

Whole Team (On-site customer)



Customer is a member of the team

- Real customer will use the finished system

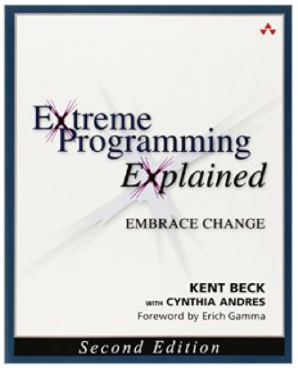
Programmers need to ask questions of a real customer

- Clarify requirements or explain what's needed

Customer sits with the team

- Customer can get some other work done while sitting with programmers

Coding standards

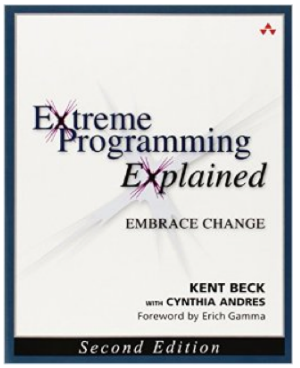


Everyone edits everyone's code

Standard should require least amount of overhead

Standard should be adopted voluntarily by the team

Workspace



http://study.com/cimages/multimages/16/openworkspace.png

Questions?