SSW-555: Agile Methods for Software Development

RUP and XP

Instructor: Prof. Zhongyuan Yu

School of Systems and Enterprises

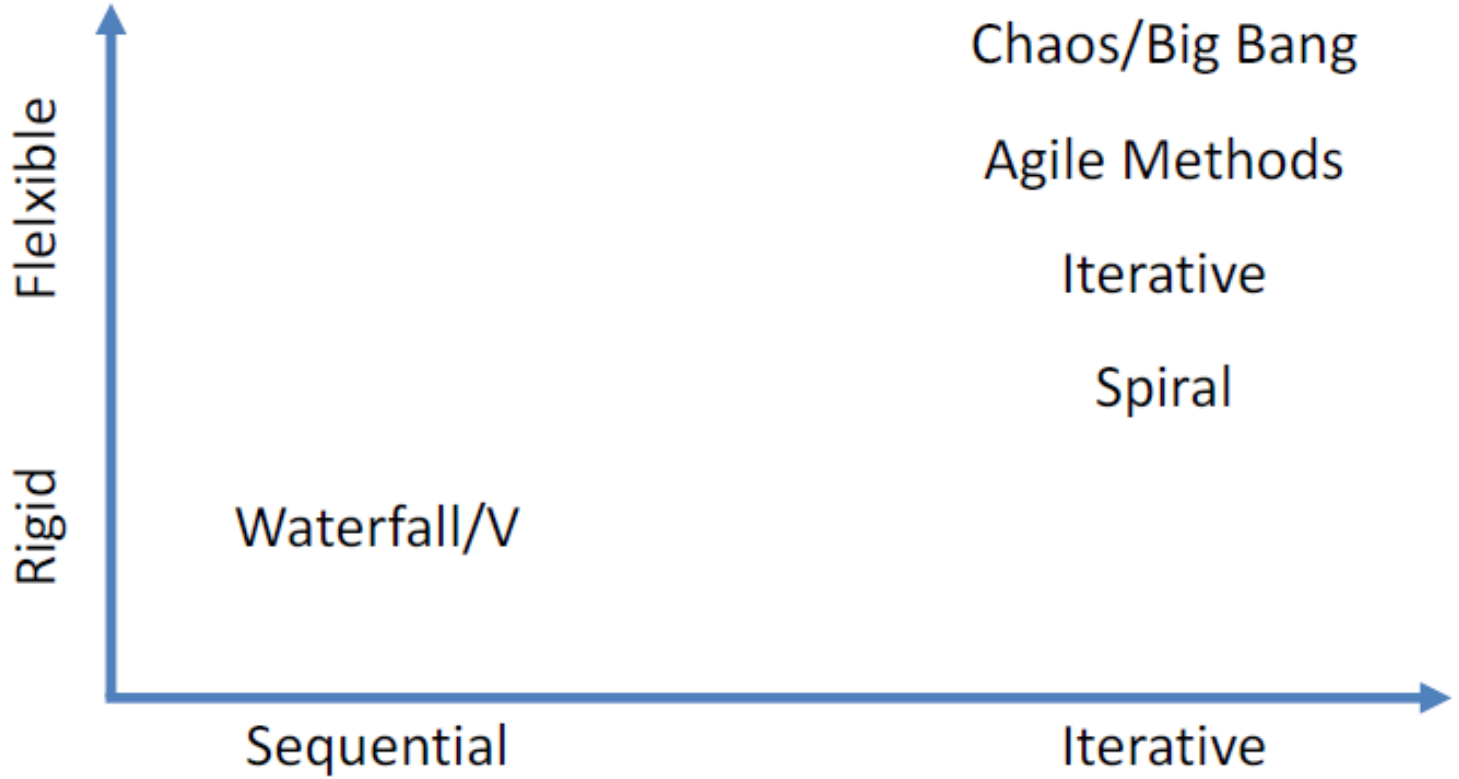
Stevens Institute of Technology

This Lecture is based on material from Prof. Jim Rowland

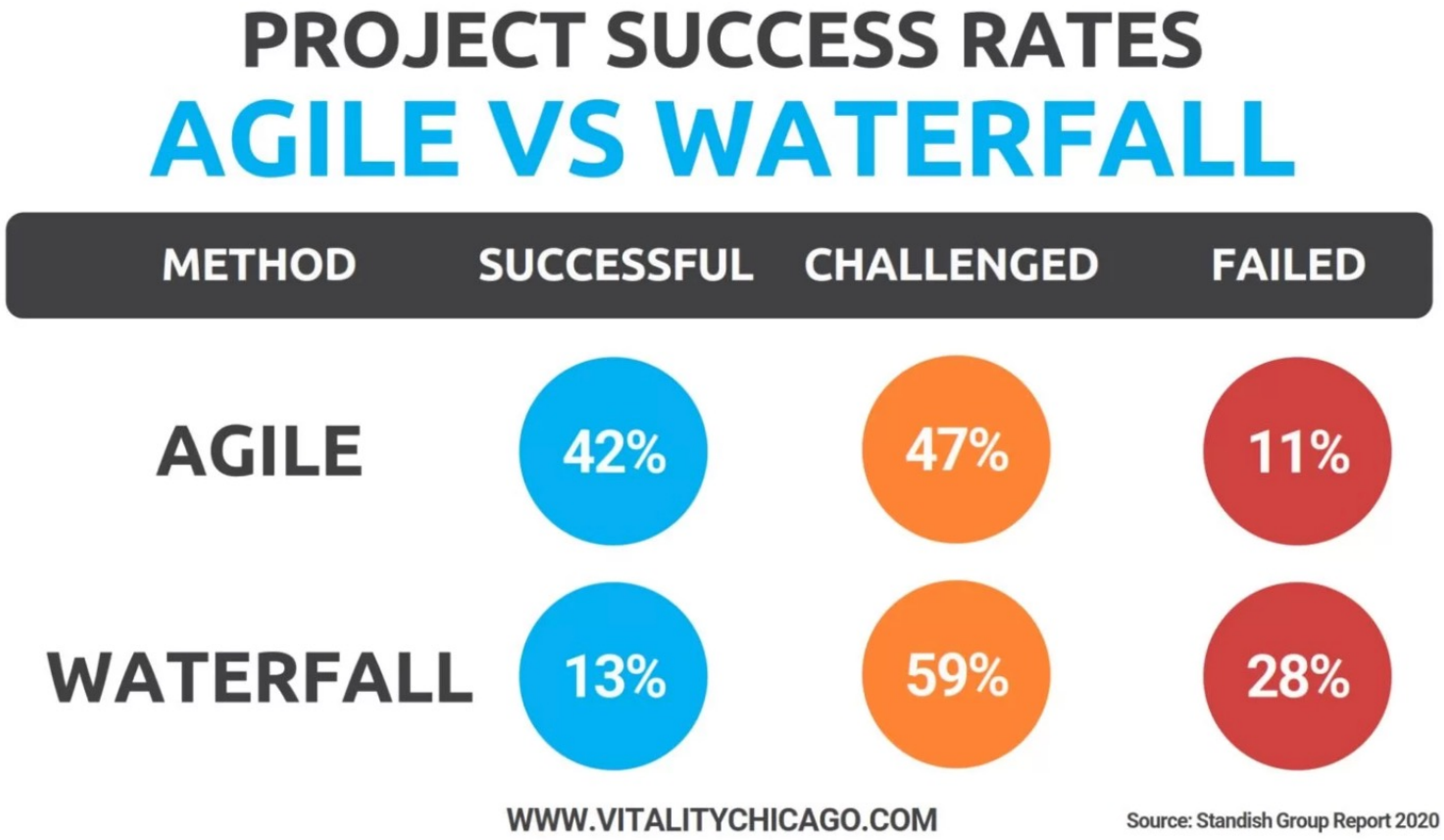
Today's Topic

- (Review) Software development method comparison
- Rational Unified Process (RUP)
- Boehm's risk exposure comparison
- Overview of Extreme Programming (XP)

SDLC Methods



Comparing Software Development Paradigms: 2020



Chaos 2020 Beyond Infinity. This report is based on an impressive database of 50,000 projects.

Today's Topic

- (Review) Software development method comparison
- **Rational Unified Process (RUP)**
- Boehm's risk exposure comparison
- Overview of Extreme Programming (XP)

Rational Unified Process (RUP)

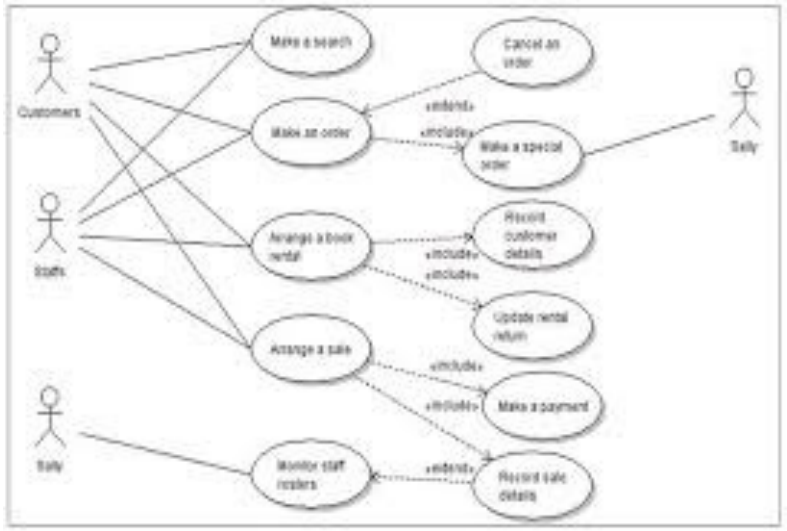
- Developed at Rational Software in late 1990s after acquisition of several Object-Oriented companies
- Based on 6 ***Best Practices*** of Software Engineering
  1. Develop iteratively
  2. Manage requirements
  3. Use component-based architectures
  4. Model software visually (UML)
  5. Continuously verify software quality
  6. Control changes

Best RUP Practices (1)

- **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

Best RUP Practices (2)

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

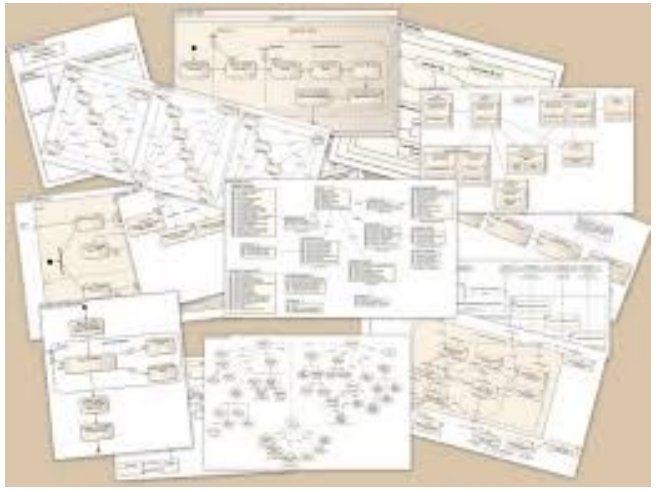


Best RUP Practices (3)

- **Use component-based architecture**
  - Focusing on early development and baselining of a robust architecture prior to full-scale development
  - Architecture should be flexible to accommodate change
  - Focus on reusable, component-based software

Best RUP Practices (4)

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



Best RUP Practices (5)

- **Verify software quality**
  - Verification and Validation is part of the process, not an afterthought
  - Focus on reliability, functionality, and performance

Best RUP Practices (6)

- **Control changes to software**
  - Change is inevitable
  - Actively manage the change request process
  - Control, track, and monitor changes

RUP Phases

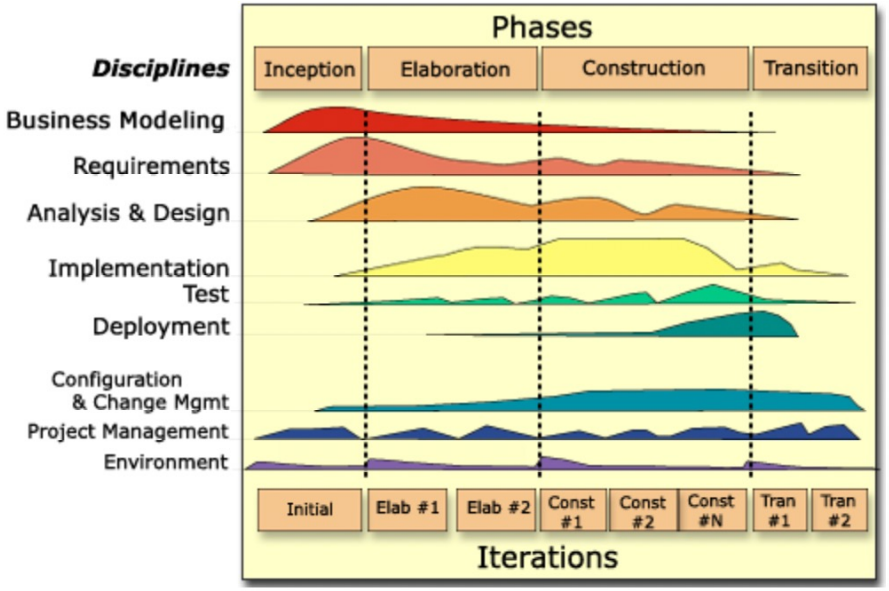
Inception: scope system for cost and budget, create basic use case model

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
- Stakeholders review progress and make go/no-go decisions



Rup Disciplines

Business Modeling: create and maintain traceability between business and software modules

Requirements: describe what the system should do

Analysis and Design: show how the system will be realized in the implementation phase

Implementation: the system is realized through implementation of reusable components

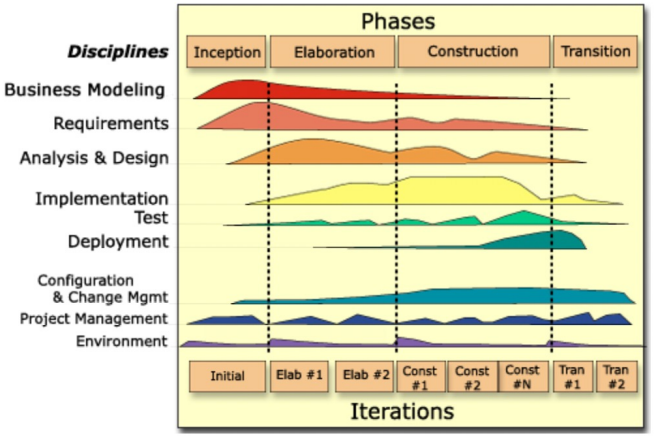
Test: find defects as early as possible

Deployment: produce product release and deliver to users

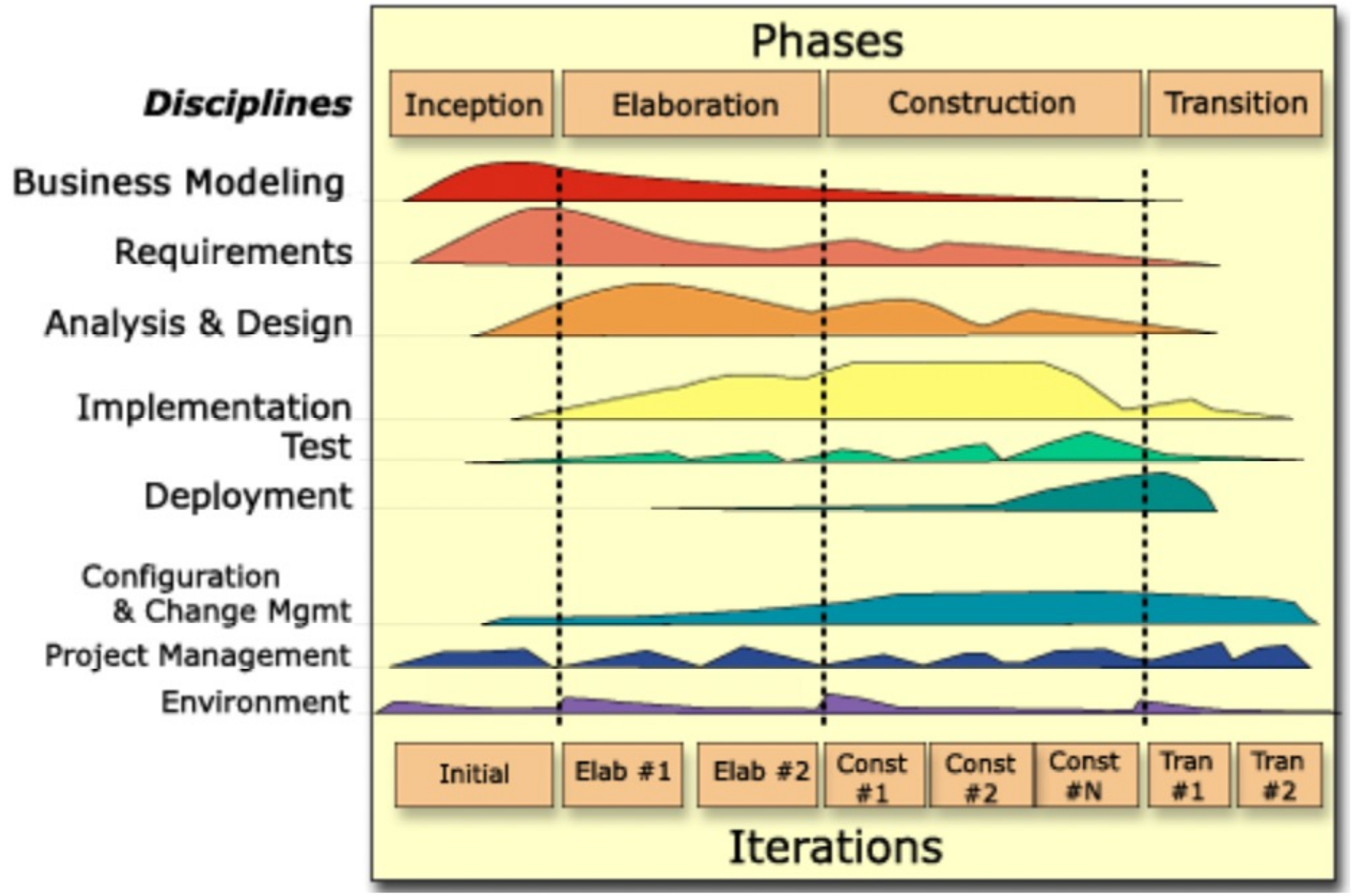
Configuration and Change Management :manage access to project work products

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

Environment: ensure that process, guidance and tools are available



RUP Phases, Iterations and Disciplines



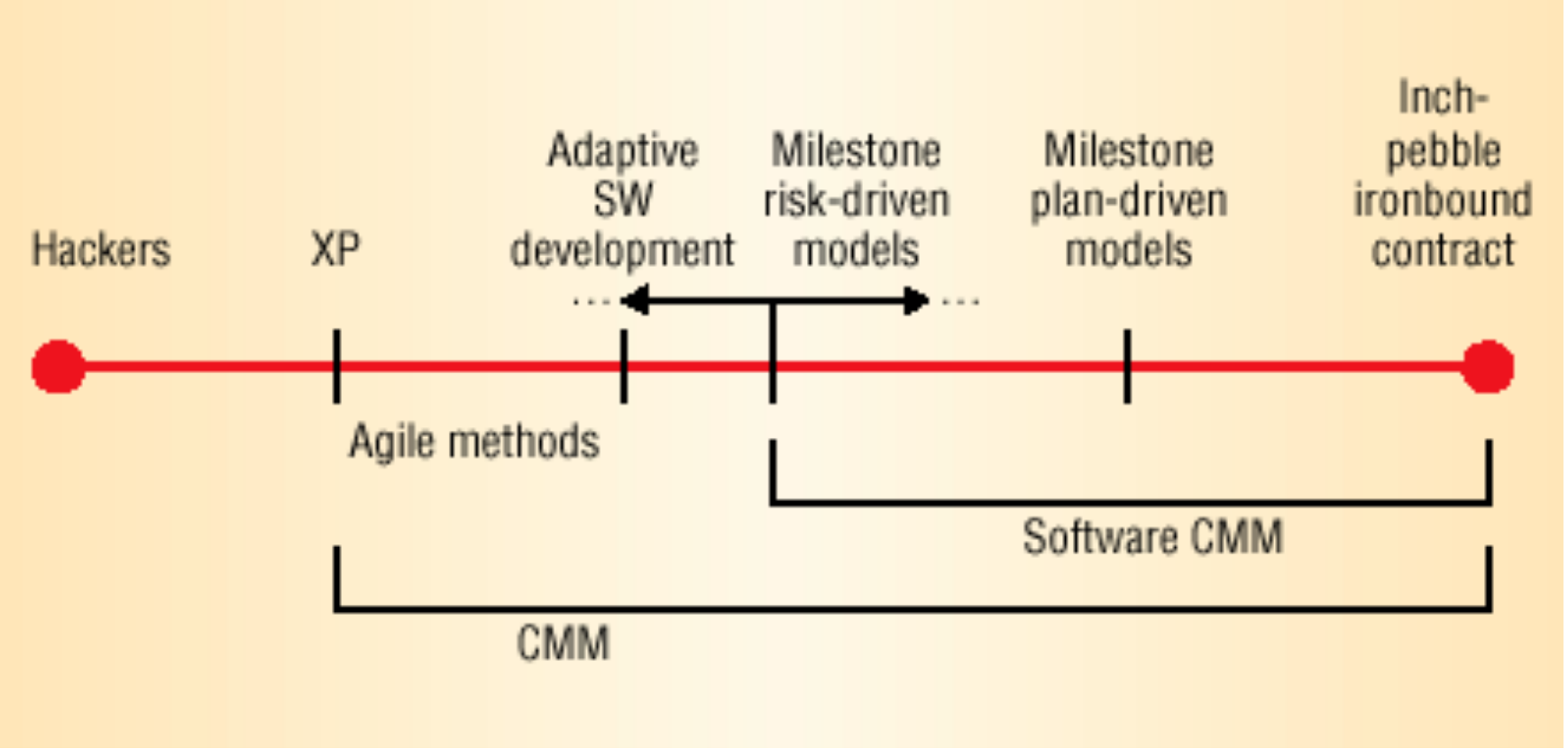
Explore more details: https://sceweb.uhcl.edu/helm/RationalUnifiedProcess/

Original reference: https://www.ibm.com/support/pages/rational-unified-process-rup-plug-ins-rational-method-composer-751

Today's Topic

- (Review) Software development method comparison
- Rational Unified Process (RUP)
- **Boehm's risk exposure comparison**
- Overview of Extreme Programming (XP)

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:

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

Black curve:
Inadequate plans

Red curve:
Loss of market share

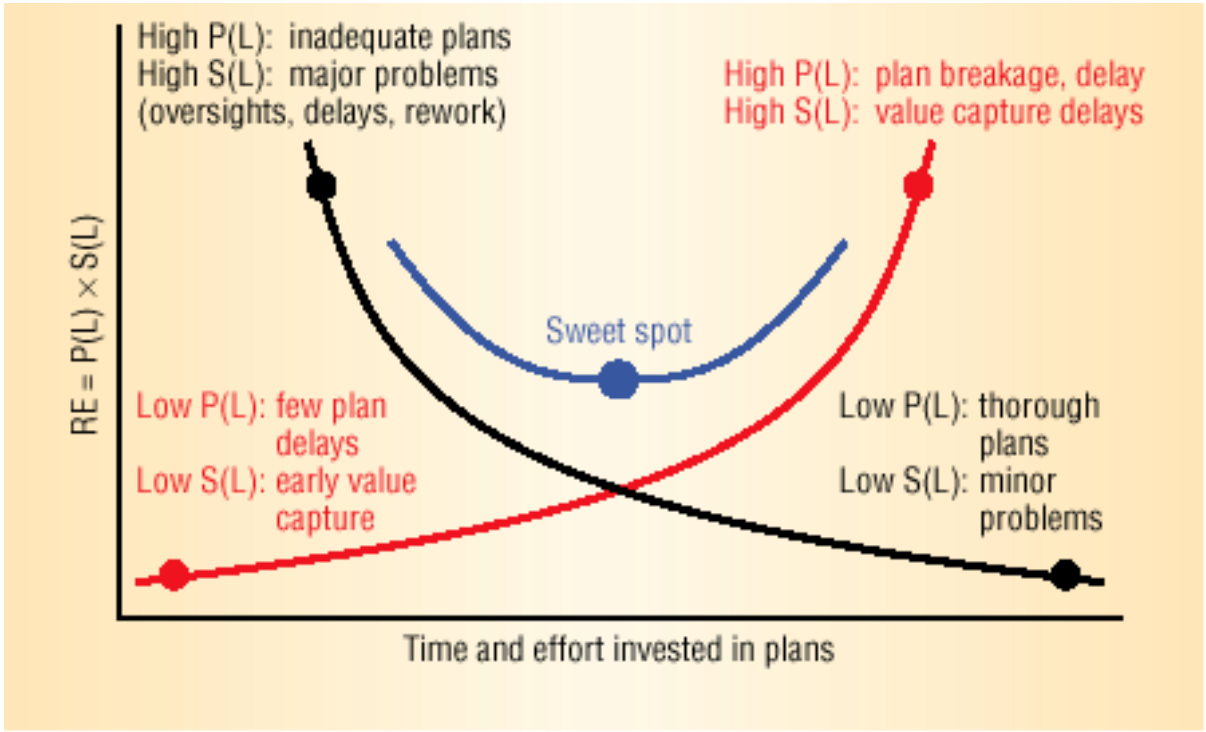


Figure 2. Risk exposure (RE) profile. This planning detail for a sample e-services company shows the probability of loss P(L) and size of loss S(L) for several significant factors.

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

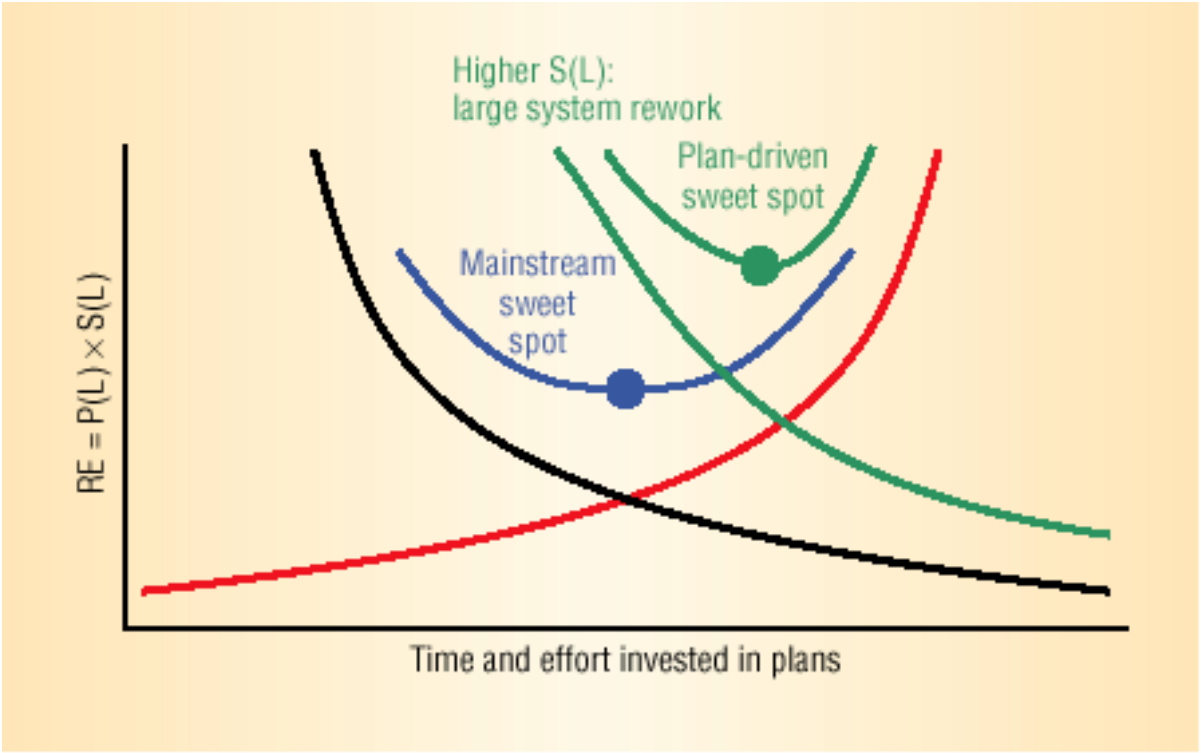


Figure 4. Comparative RE profile for a plan-driven home-ground company that produces large, safety-critical systems.

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

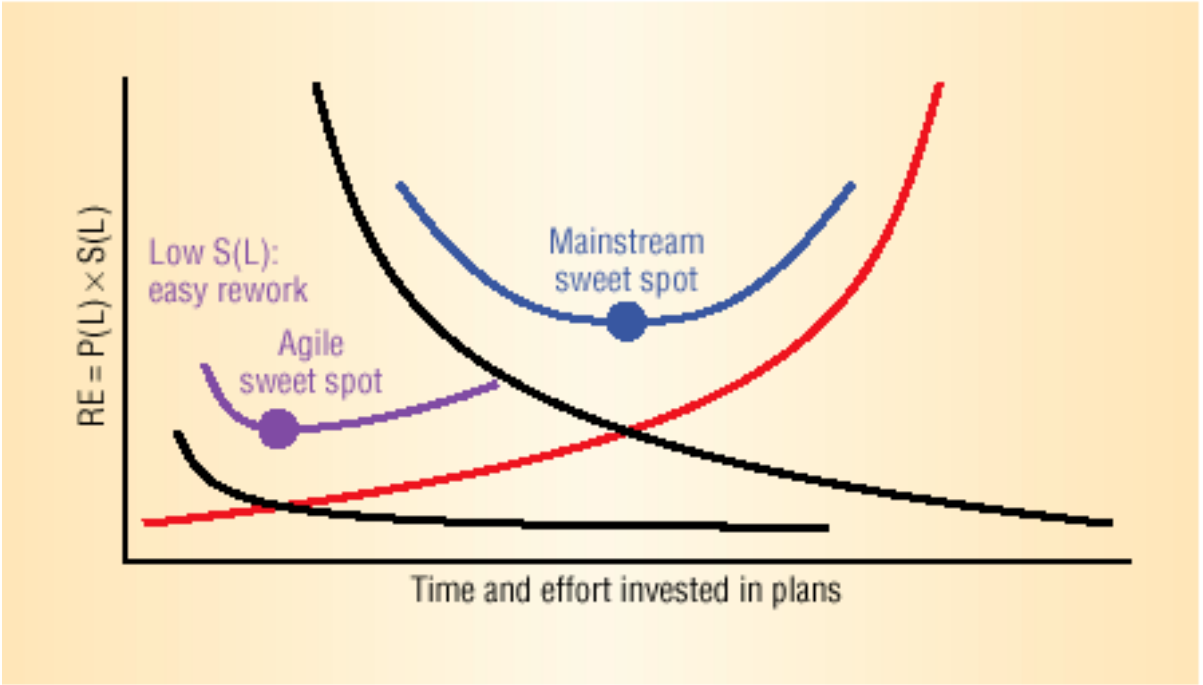


Figure 3. Comparative RE profile for an agile home-ground company with a small installed base and less need for high assurance.

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

Agile Manifesto (2001)

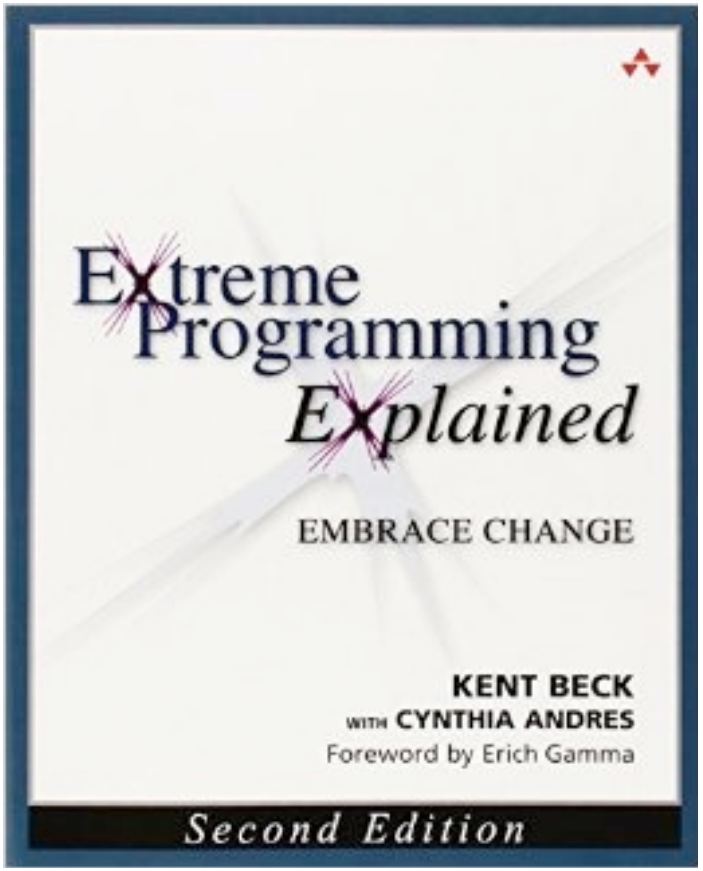
12 Principles behind the Agile Manifesto
http://agilemanifesto.org/

Today's Topic

- (Review) Software development method comparison
- Rational Unified Process (RUP)
- Boehm's risk exposure comparison
- **Overview of Extreme Programming (XP)**

Extreme Programming (XP)

- Created by Kent Beck while working on a project for Chrysler in the late 1990s with collaborators: Ward Cunningham and Ron Jeffries
- One of the most well-known agile methods
- Takes best practices to extreme levels
- http://www.extremeprogramming.org/



12 Extreme Programming Practices

Group	Practices
Feedback	• The Planning Game • Whole team • Pair programming • Testing
Continuous Process	• Continuous integration • Small releases • Refactoring
Code	• Simple design • Coding standards • Collective ownership • System Metaphor
Work conditions	• Sustainable pace

1. The Planning Game

The main planning process within extreme programming is called the Planning Game. The game is a meeting that occurs once per iteration, typically once a week.

Businesspeople decide:
- scope
- priority
- release dates

Technical people decide:
- estimates of effort
- technical consequences
- process
- detailed scheduling

The Planning Game has the following advantages:

✓ Reduction in time wasted on useless features

✓ Greater customer appreciation of the cost of a feature

✓ Less guesswork in planning

2. Whole Team

- 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

3. 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
- ✓ Two heads are better than one

4. Testing

- The developers continually write unit tests, which need to pass for the development to continue.
- The customers write tests to verify that the features are implemented.
- The tests are automated so that they become a part of the system and can be continuously run to ensure the working of the system.

The advantages of testing are:

- ✓ Unit testing promotes testing completeness
- ✓ Test-first gives developers a goal
- ✓ Automation gives a suite of regression tests

5. 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

The advantages:

✓ Reduces the duration, which is otherwise lengthy.

✓ Enables the short releases practice as the time required before release is minimal.

6. 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

The advantages of Short Releases are:

✓ Frequent feedback

✓ Tracking

✓ Reduce chance of overall project slippage

7. Refactoring

Developers restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility. This is called Refactoring.

- The developers ask if they can see how to make the code simpler, while still running all the tests.

The advantages of Refactoring are:

✓ It become easier to make the next changes

✓ Increases the developer knowledge of the system

8. Simple Design

- Runs all the tests
- Has no duplicated logic like parallel class hierarchies
- States every intention important to the developers
- Has the fewest possible classes and methods

The advantages of Simple Design are:

- ✓ Time is not wasted adding superfluous functionality
- ✓ Easier to understand what is going on
- ✓ Refactoring and collective ownership is made possible
- ✓ Helps keep the programmers on track

9. Coding Standard

- Communication through the code.
- The least amount of overhead.
- Voluntary adoption by the whole team.

The advantages :

- ✓ Supports collective ownership
- ✓ Reduces the amount of time developers spend reformatting other peoples' code
- ✓ Reduces the need for internal commenting
- ✓ Calls for clear, unambiguous code

10. Collective Ownership

- The entire team takes responsibility for the whole of the system.
- Not everyone knows every part equally well, but everyone knows something about every part.
- If developers see an opportunity to improve the code, they go ahead and improve it.

The advantages:

- ✓ Helps mitigate the loss of a team member who is leaving.
- ✓ Promotes the developers to take responsibility for the system as a whole rather than parts of the system.

11. System Metaphor

- Metaphor is a simple explanation of the project
  - Agreed upon by all members of the team
  - Simple enough for customers to understand
  - Detailed enough to drive the architecture

The advantages of Metaphor are:

- ✓ Encourages a common set of terms for the system
- ✓ Reduction of buzz words and jargon
- ✓ A quick and easy way to explain the system

12. Sustainable Pace

- 40 hours per week: most developers lose effectiveness past 40 hours.
- Overtime is a symptom of a serious problem
- XP only allows one week of overtime

The advantages:

- ✓ People should be fresh and eager every morning
- ✓ Value is placed on the developers' well-being.
- ✓ Management is forced to find real solutions.

Workspace

STEVENS
INSTITUTE OF TECHNOLOGY
1870

THANK **YOU**

Stevens Institute of Technology
1 Castle Point Terrace, Hoboken, NJ 07030