
STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

# SSW-555: Agile Methods for Software Development

*Introduction Extreme Programming/XP*

# 2018 Fall Office Hour Update

**Thursday 1:00-5:00 pm**

**Babbio 513**

**lxiao6@stevens.edu**

**201 216 3676**

# 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/
- http://www.tutorialspoint.com/extreme_programming/
- https://www.infoq.com/articles/reifer-agile-study-2017

# Agenda

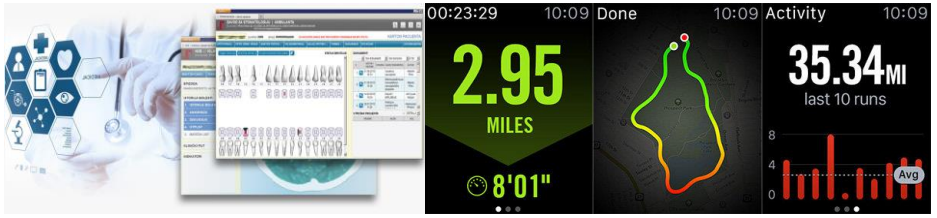
- Software development method comparison
- Motivations for agile methods
  - Rational Unified Process (RUP)
  - Boehm's risk exposure comparison
- Overview of Extreme Programming (XP)

# Agenda

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

# 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?

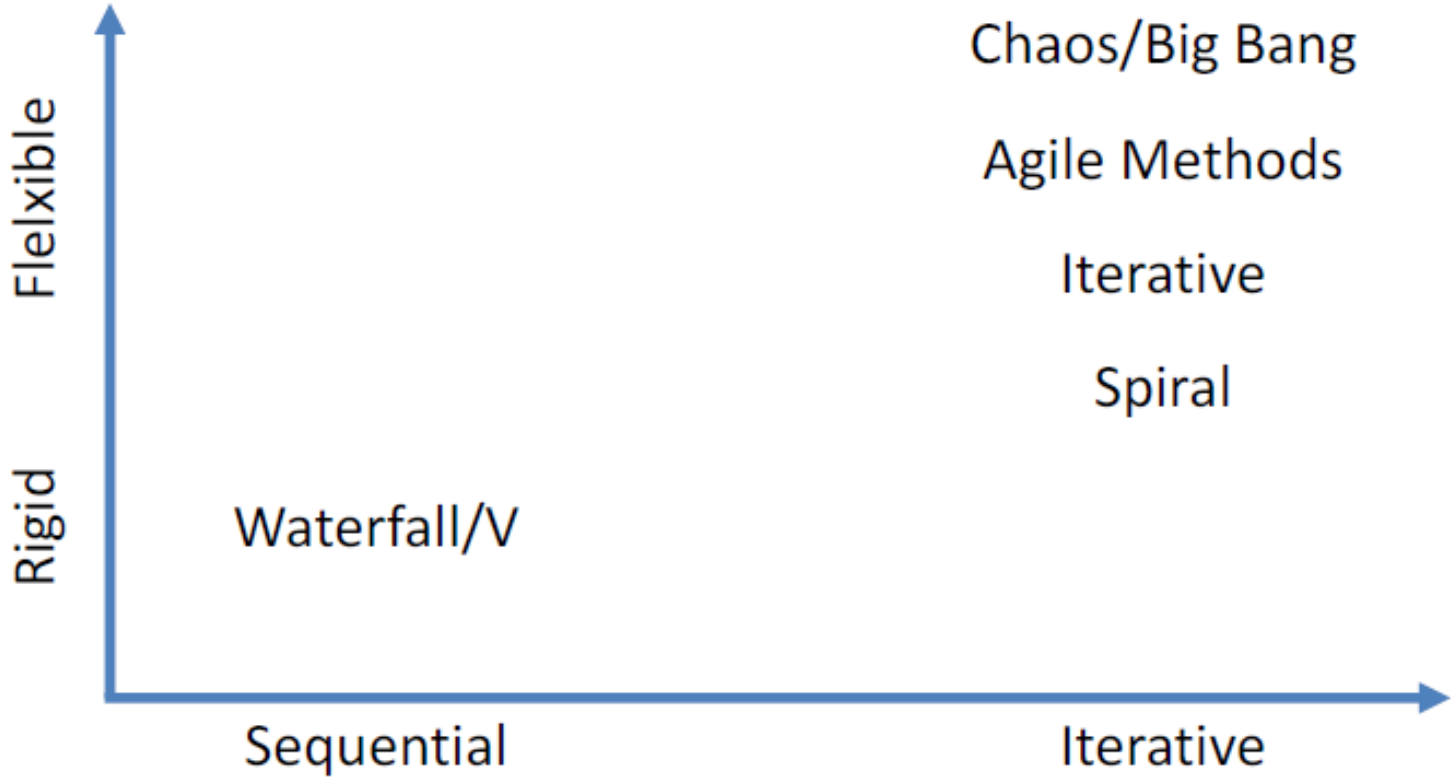


# Software Development Life Cycle (SDLC)

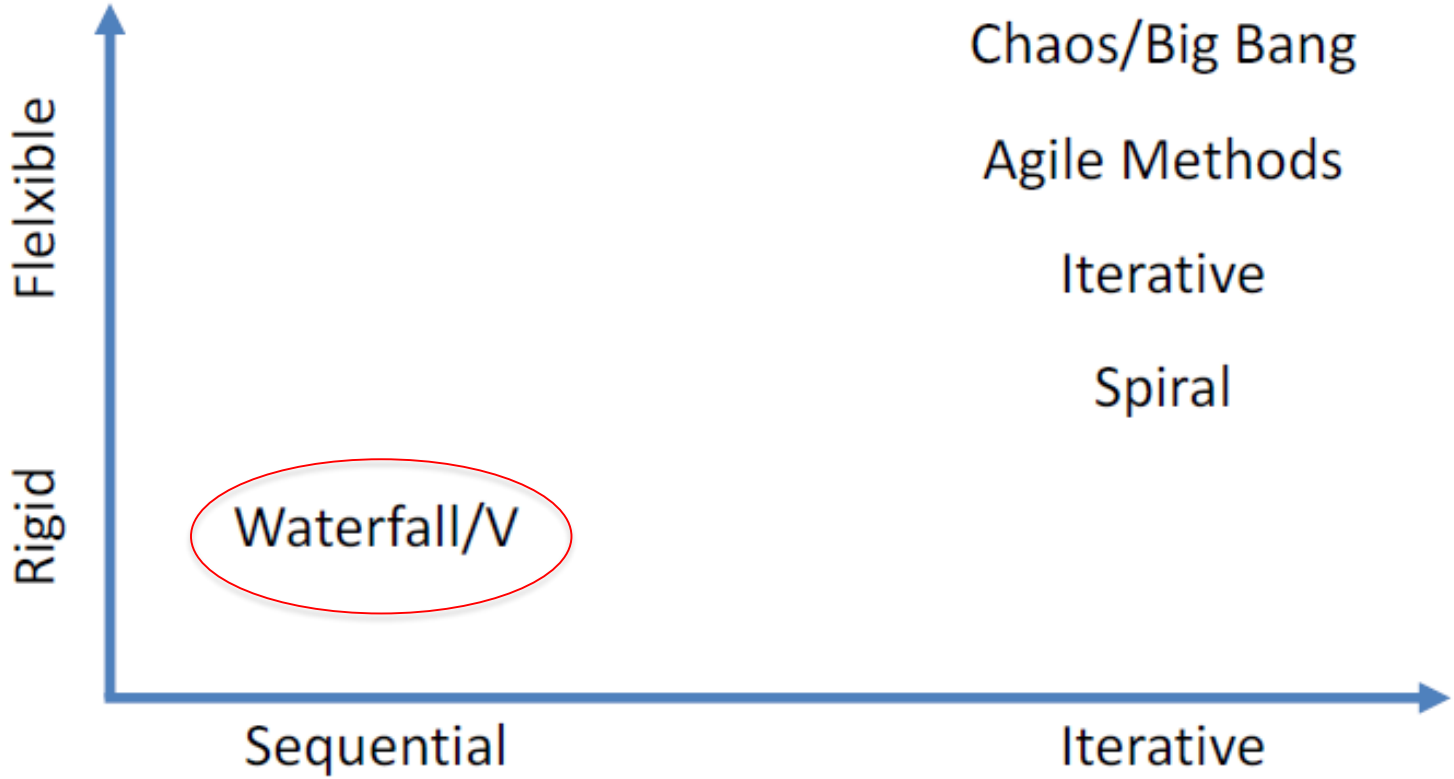
| | |
|---|---|
| Software specification | • What functionality must we support? |
| Software development | • How do we create the software that delivers the functionality? |
| Software validation | • How do we verify that the software does what it's supposed to do? |
| Software evolution | • How does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

Source: Software Engineering, 10th Edition, Ian Sommerville

# SDLC Methods



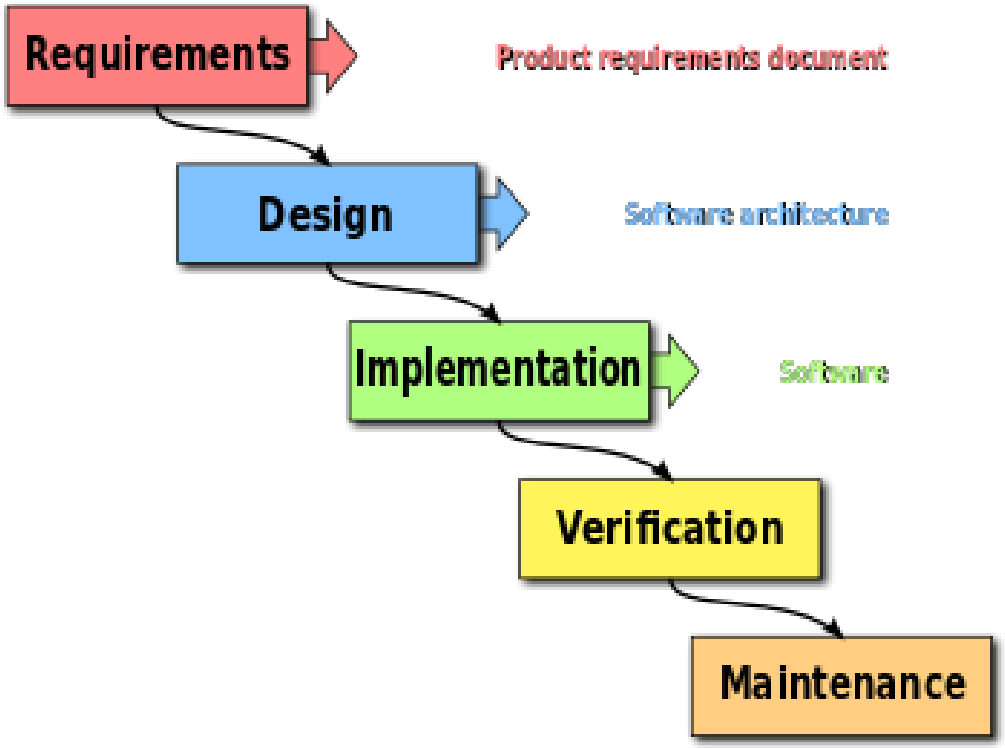
# SDLC Methods



# Meet the Famous Waterfall Model

A sequential (non-**iterative**) design process, seen as flowing steadily downwards (like a **waterfall**).

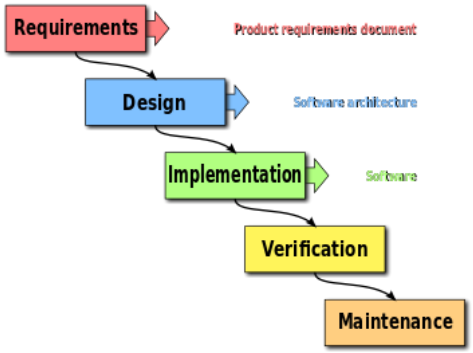
# Meet the Famous Waterfall Model




A sequential (non-**iterative**) design process, seen as flowing steadily downwards (like a **waterfall**) through the different phases.

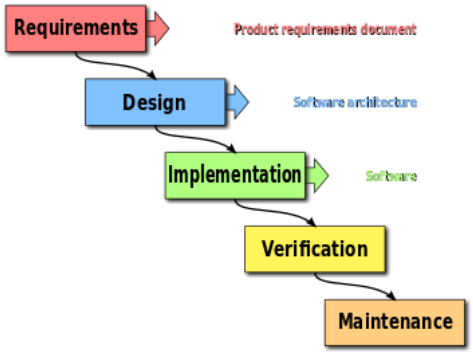
# Meet the Famous Waterfall Model

- Sufficient planning
- Popular with traditional engineering problems, e.g. building a bridge
- Very formal process
  - Extensive documentation
  - Serial execution
  - Strict paper work “signed off in blood”



# Meet the Famous Waterfall Model

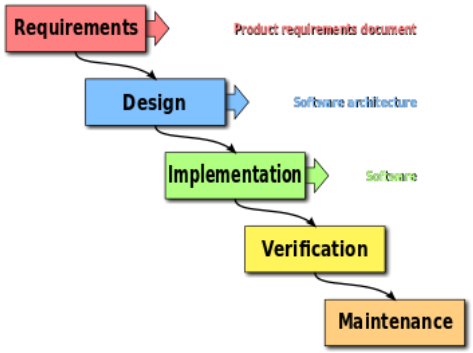
- When is waterfall appropriate?



# Meet the Famous Waterfall Model

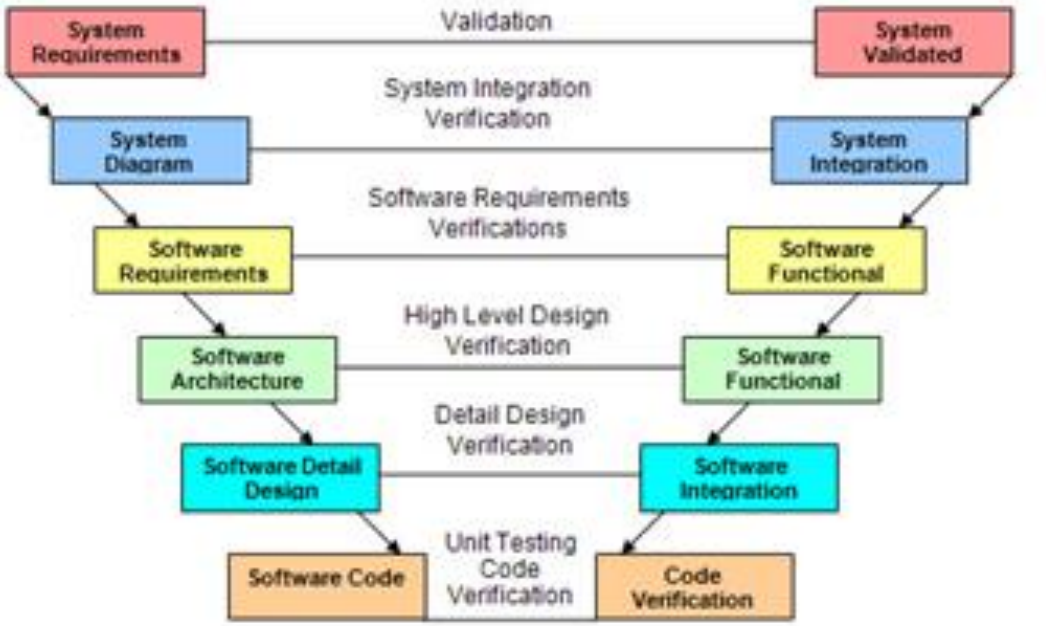
When is waterfall appropriate?

- Must get it right the first time!
  - Interacting with hardware systems, which are difficult to change
- Cost of failure is very high
  - Critical systems with safety or security requirements, e.g. airplanes, self-driving cars…
- Complete and stable requirements
  - Very large, multi-organizational software projects



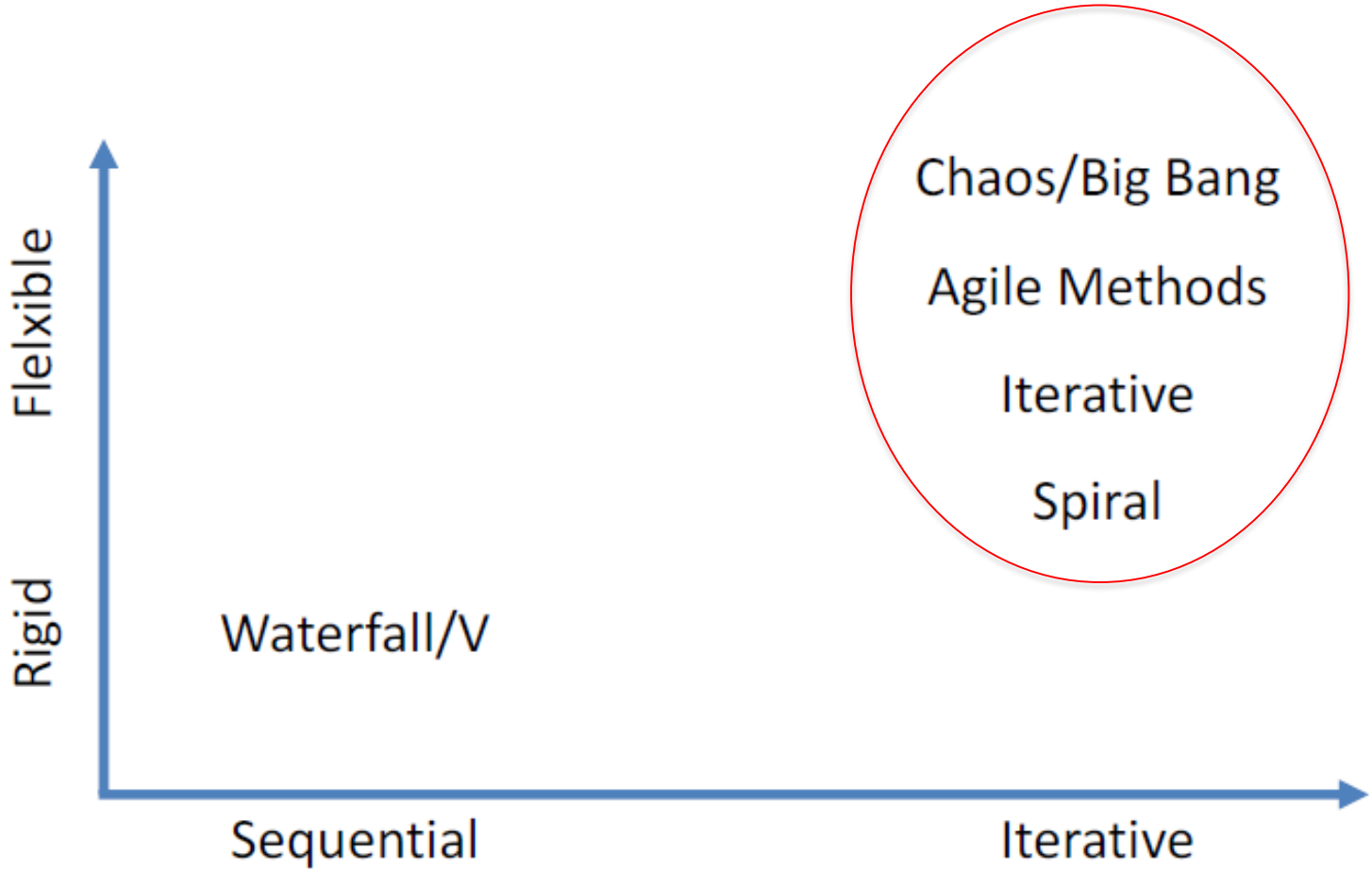
# V (Verification/Validation) SDLC Model

- Add verification/testing to each step of the Waterfall Model
- Must complete rigorous testing before proceeding to next step



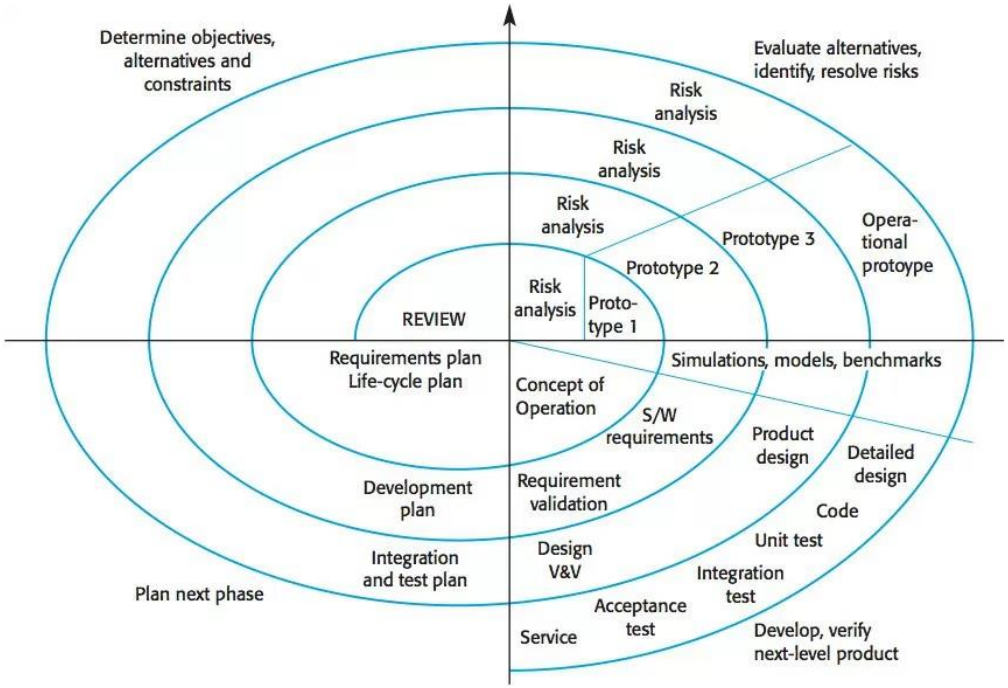

https://sites.google.com/site/advancedsofteng/softwareacquisition/software-development-lifecycle-approaches

# SDLC Methods



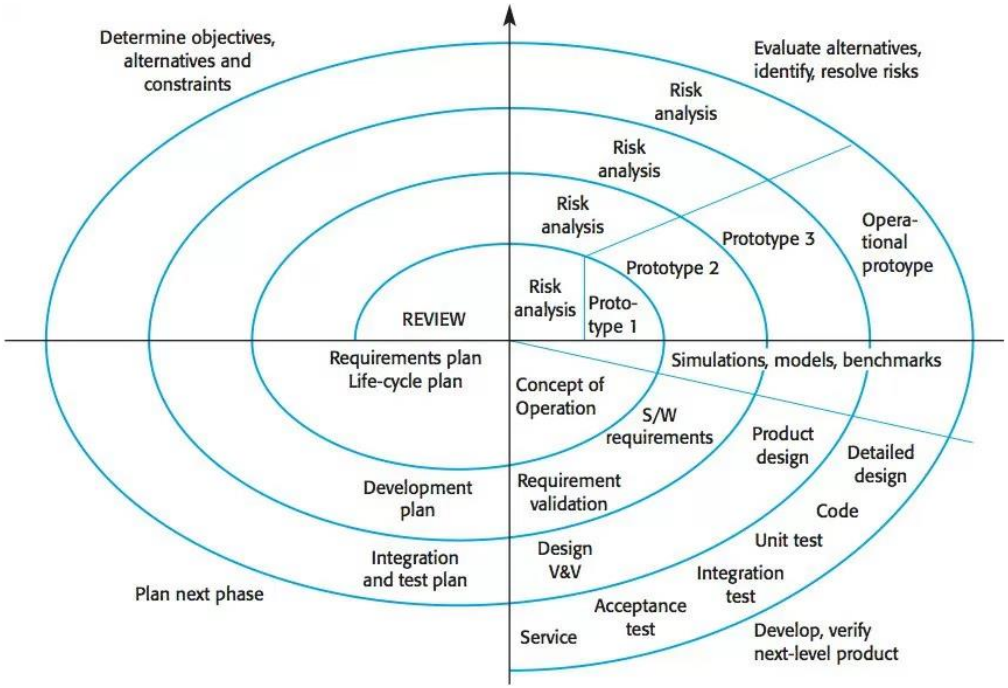
# Boehm's Spiral Model

- Limitations of waterfall?




http://csse.usc.edu/TECHRPTS/1988/usccse88-500/usccse88-500.pdf

# Boehm's Spiral Model

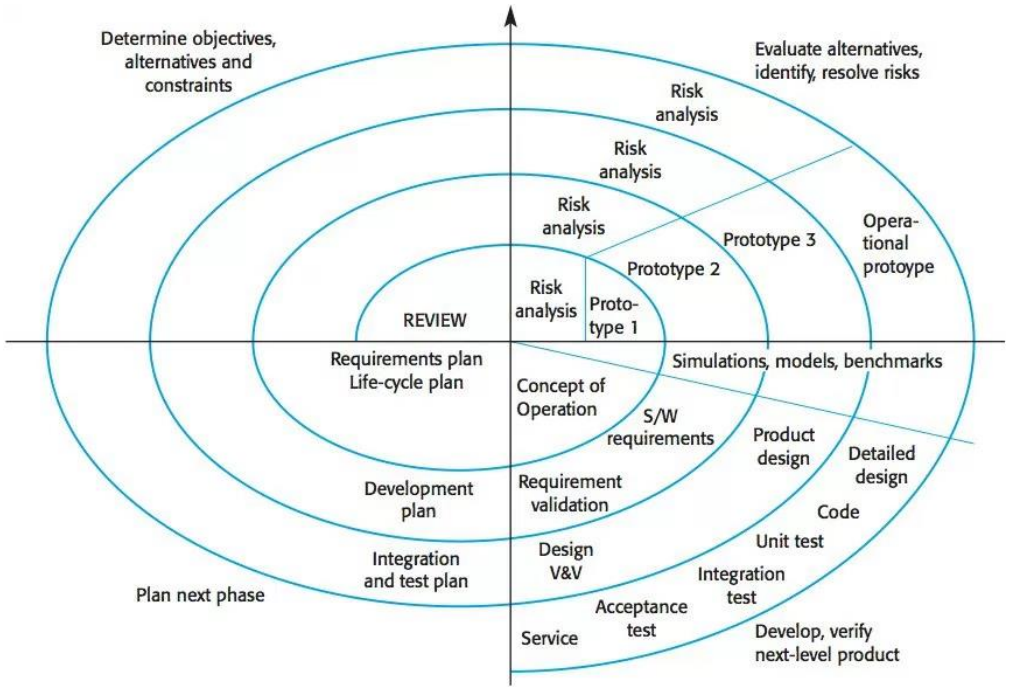



http://csse.usc.edu/TECHRPTS/1988/usccse88-500/usccse88-500.pdf

- Limitations of waterfall?

Hard to get it right the first time and changing requirements…

# Boehm's Spiral Model



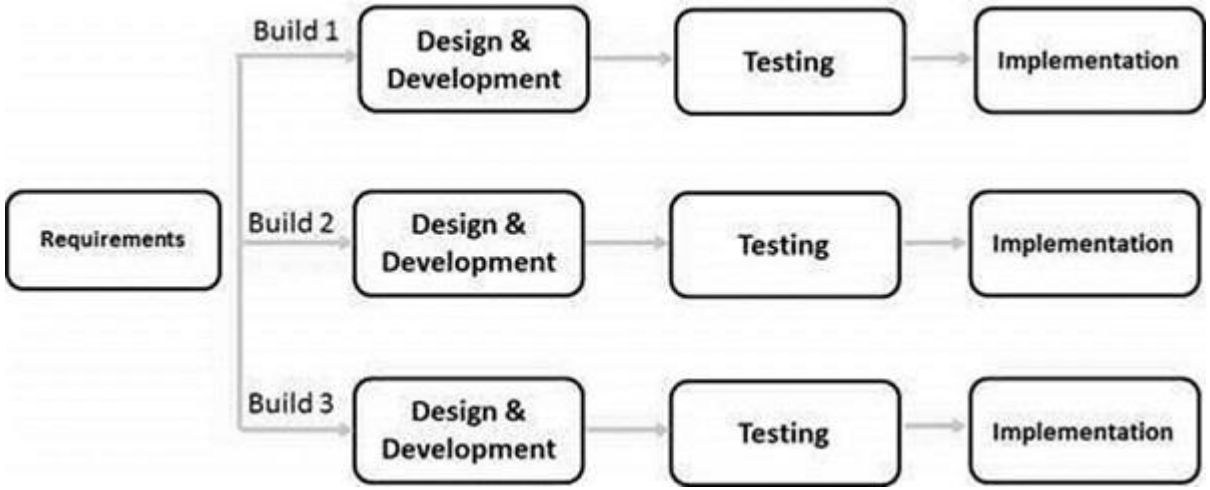

http://iansommerville.com/software-engineeringbook/web/spiral-model/

- Limitations of waterfall?

Hard to get it right the first time and changing requirements…

- Incremental development and interactions
  1. Objective setting
  2. Risk assessment and reduction
  3. Development and validation
  4. Planning for next iteration

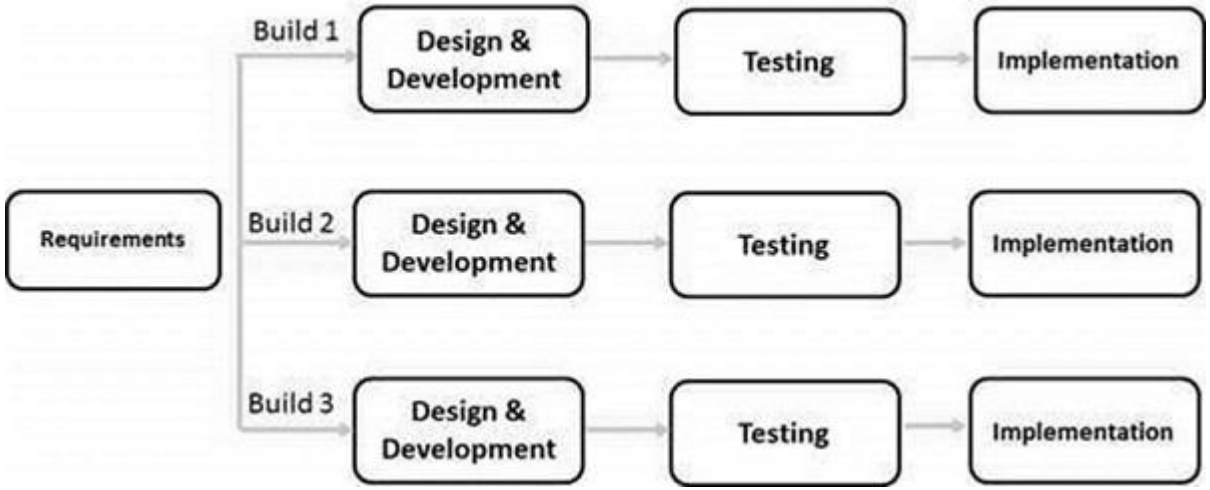
# Iterative models SDLC




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

- Identify requirements up front
  - Mostly stable but may change…
- Build subsets of requirements
  - Sequentially or in parallel with multiple groups of developers

# Iterative models SDLC

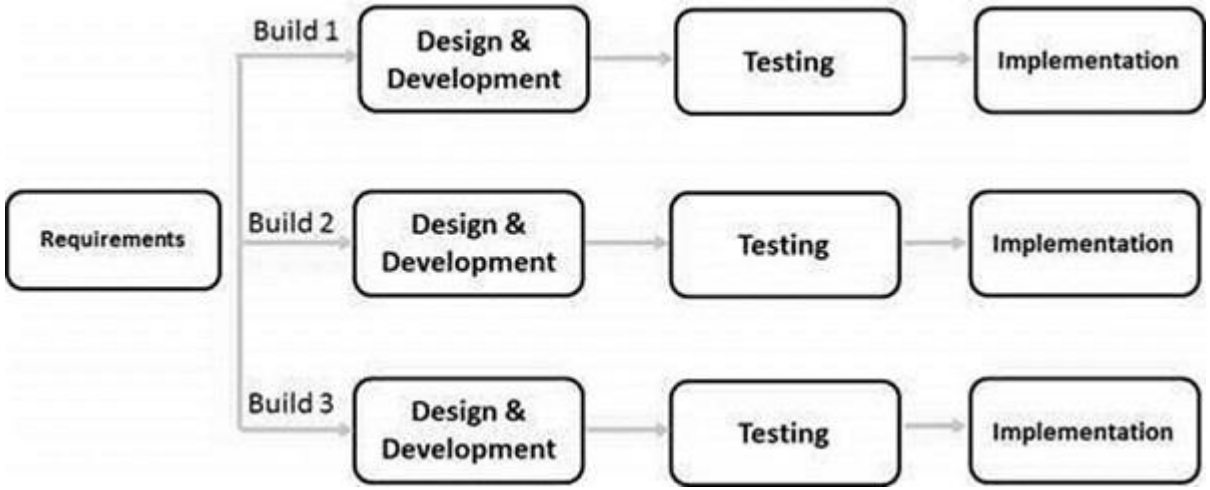



Benefits?

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

- Identify requirements up front
  - Mostly stable but may change…
- Build subsets of requirements
  - Sequentially or in parallel with multiple groups of developers

# Iterative models SDLC




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

Benefits?

- Parallel effort supported by multiple teams
- Delivers early functionality to customers for review

- Identify requirements up front
  - Mostly stable but may change…
- Build subsets of requirements
  - Sequentially or in parallel with multiple groups of developers

# Agile Methods SDLC


https://www.linkedin.com/pulse/what-agile-methodologydisadvantage-waterfall-model-bikesh-srivastava

"Agile software Development" is an umbrella term for frequent iterative and incremental programming development approaches.

e.g. Extreme Programming (XP), Scrum, Crystal, DSDM, Lean, FDD

# Agile Methods SDLC


https://www.linkedin.com/pulse/what-agile-methodologydisadvantage-waterfall-model-bikesh-srivastava

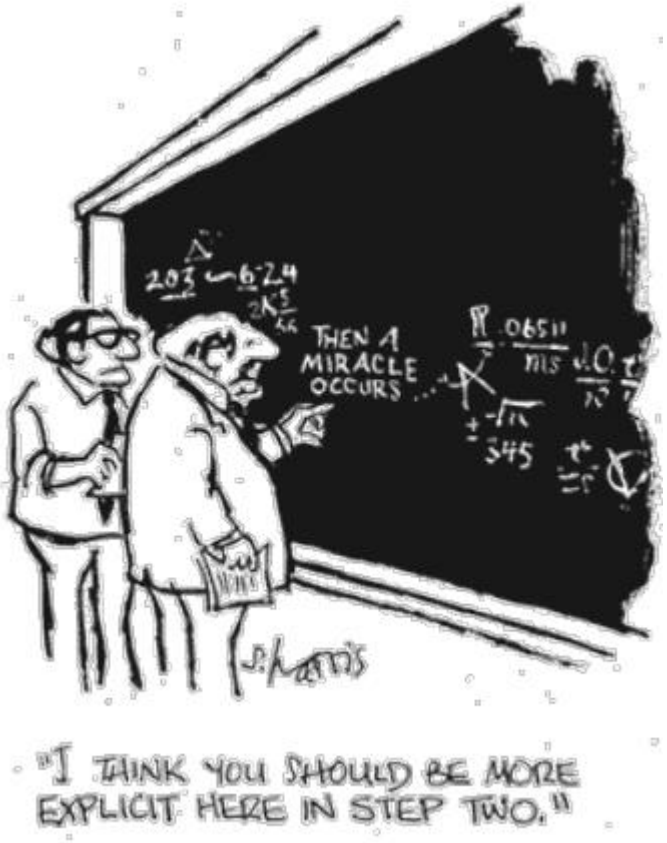
- Frequent iterations and deliverables
- Close collaboration between developers and customers
- Support changing requirements
- Frequent retrospection: learn and improve from experience

# Reasons to Use Agile Methods

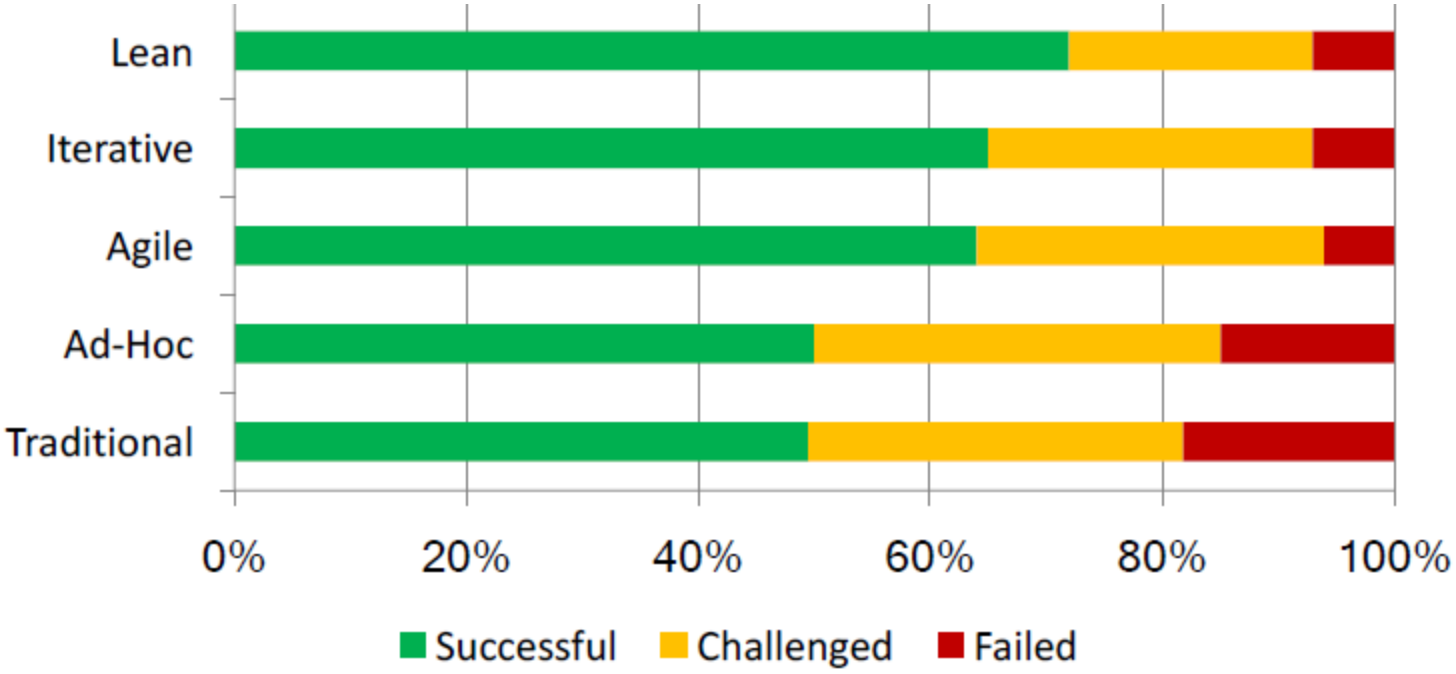
- Big, upfront planning is not practical because of unstable (changing) and ambiguous requirements
- Delivery through small baby steps through iterative and incremental development to reduce the chances of risk.
- Visibility with customers: customers are part of the team instead of being purely observers.
- Frequent reflections by the project teams
  - What are we doing well?
  - What can we improve?

# Big Bang/Chaos SDLC

- Little to no planning
- Figure it out as you go
- Typically used for very small projects (e.g. course projects…, small start-ups)
- Not highly recommended…

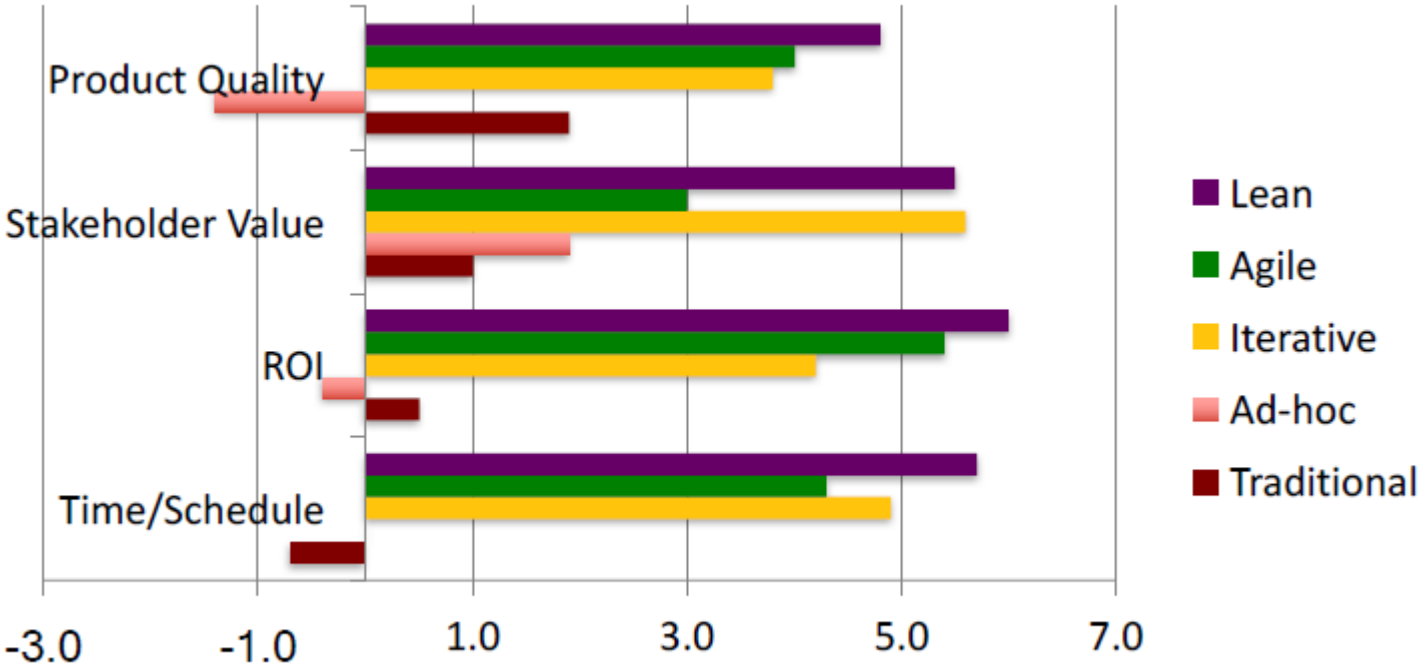


# Comparing Software Development Paradigms: 2013




SCOTT AMBLER + Associates

# Comparing Delivery Paradigms




SCOTT AMBLER + Associates

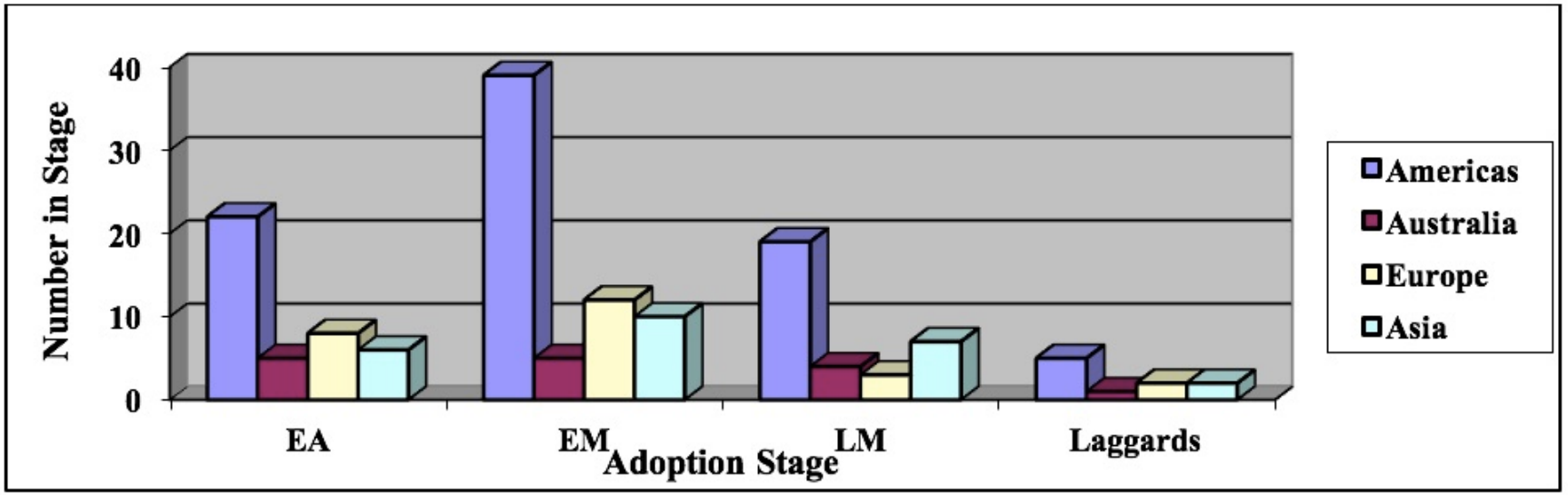
# Software Development before Agile

- 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

# 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

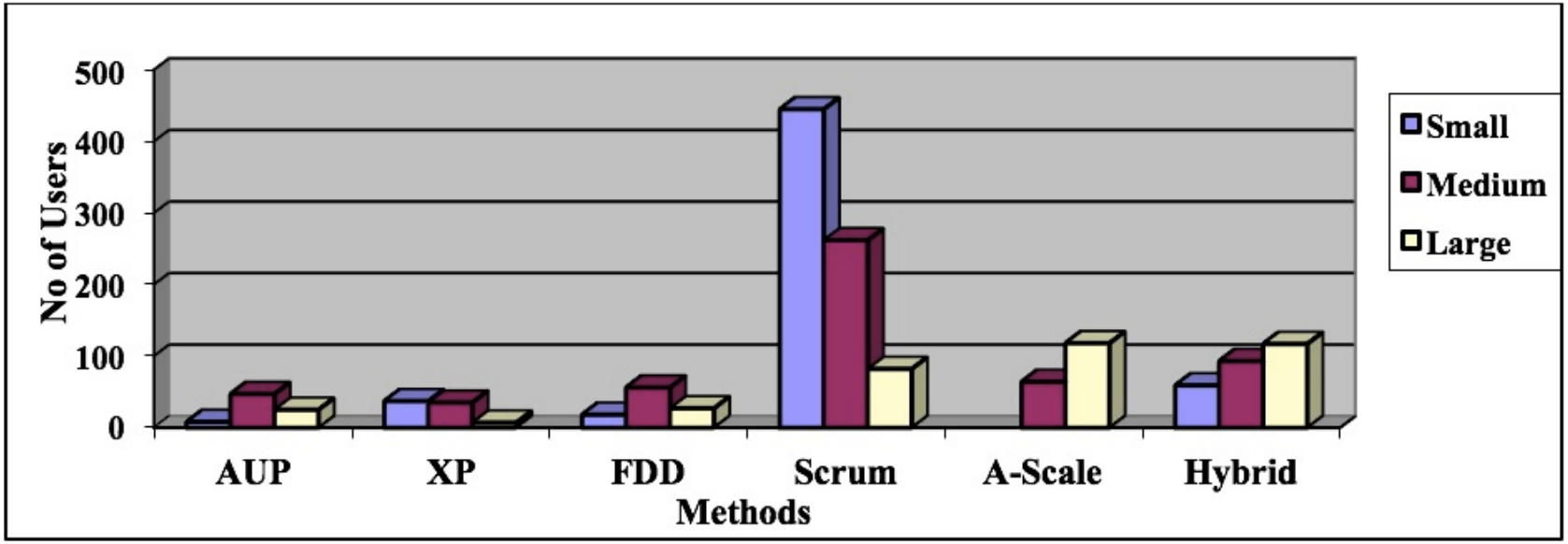
# Worldwide Agile Adoption Phase




- **EA**: Early Adopters (41) – introduced agile methods during period from 2004 to 2009.
- **EM:** Early Majority (66) – introduced agile methods during period from 2009 to 2014.
- **LM**: Late Majority (33) – introducing agile methods presently.
- **Laggards** (10) – considering, but not currently using agile methods.

https://www.infoq.com/articles/reifer-agile-study-2017

# Most Popular Agile Methodology




**Notes**

- **Small Project (567):** project that delivers a product can be developed by a single agile team.
- **Medium Projects (581):** agile project that uses 2 to 5 teams at same locations to develop products.
- **Large Projects (352)**: large project that uses 5 or more teams often at different locations to develop products.

**Legend**

- **AUP** – Agile Unified Process
- **Scrum** – Scrum and derivatives
- **XP** – Extreme Programming
- **A-Scale** – Agile at scale methods
- **FDD** – Feature-Driven Development
- **Hybrid** – Mix of methods

https://www.infoq.com/articles/reifer-agile-study-2017

# Agenda

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

# Rational Unified Process (RUP)

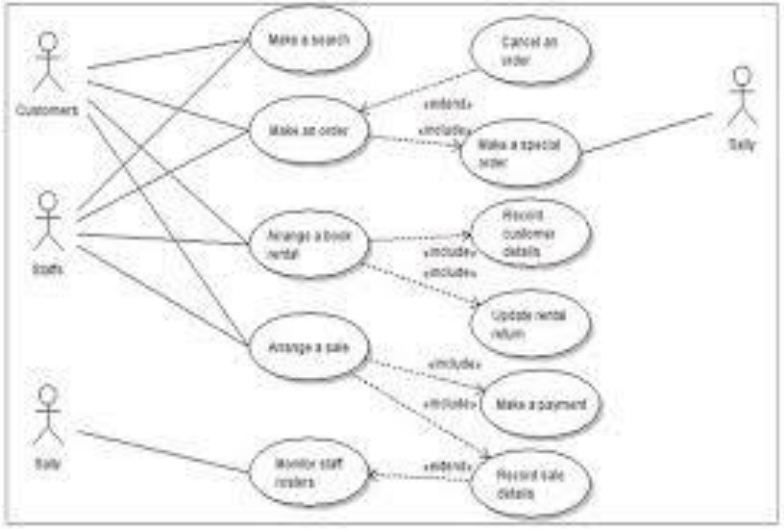
- Developed at Rational Software in late 1990s after acquisition of several Object Oriented companies
- Based on 6 ***Best Practices*** of Software Engineering
  1. Develop iteratively
  2. Manage requirements
  3. Use component-based architectures
  4. Model software visually (UML)
  5. Continuously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

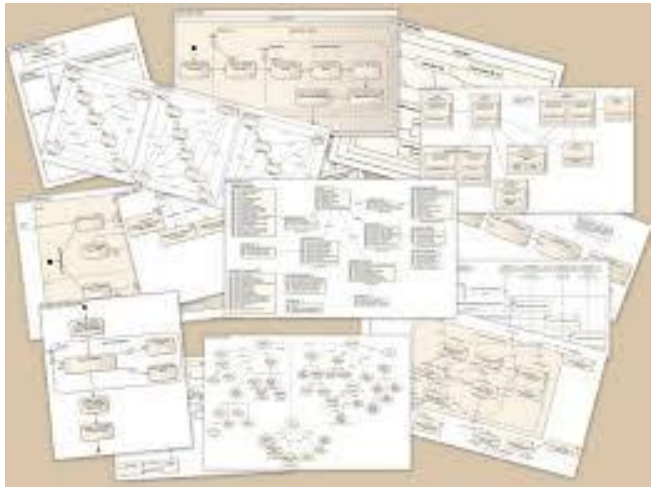


# Best RUP Practices (3)

- **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

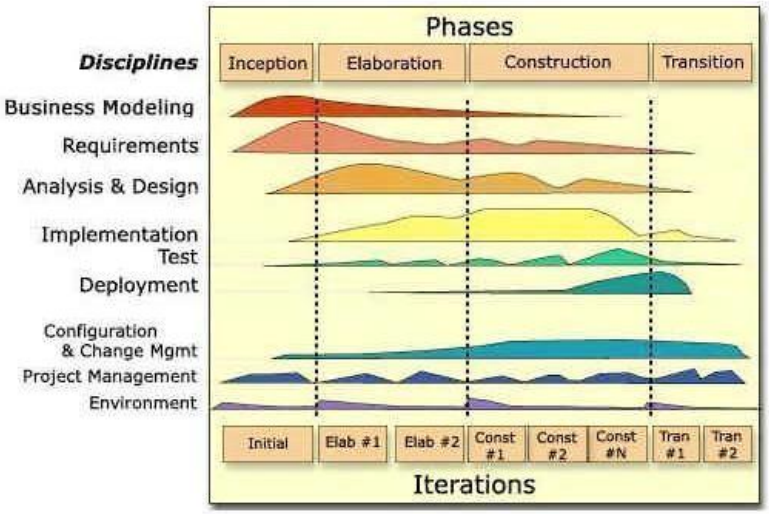
***Inception***: scope system for cost and budget, create basic use case model

***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

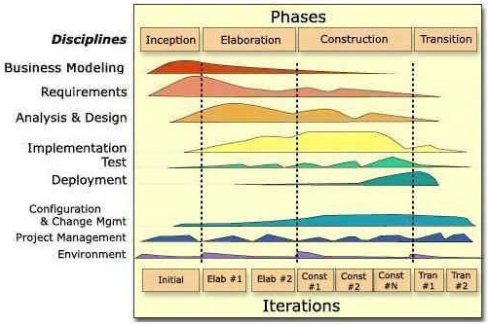
***Test***: find defects as early as possible

***Deployment***: produce product release and deliver to users

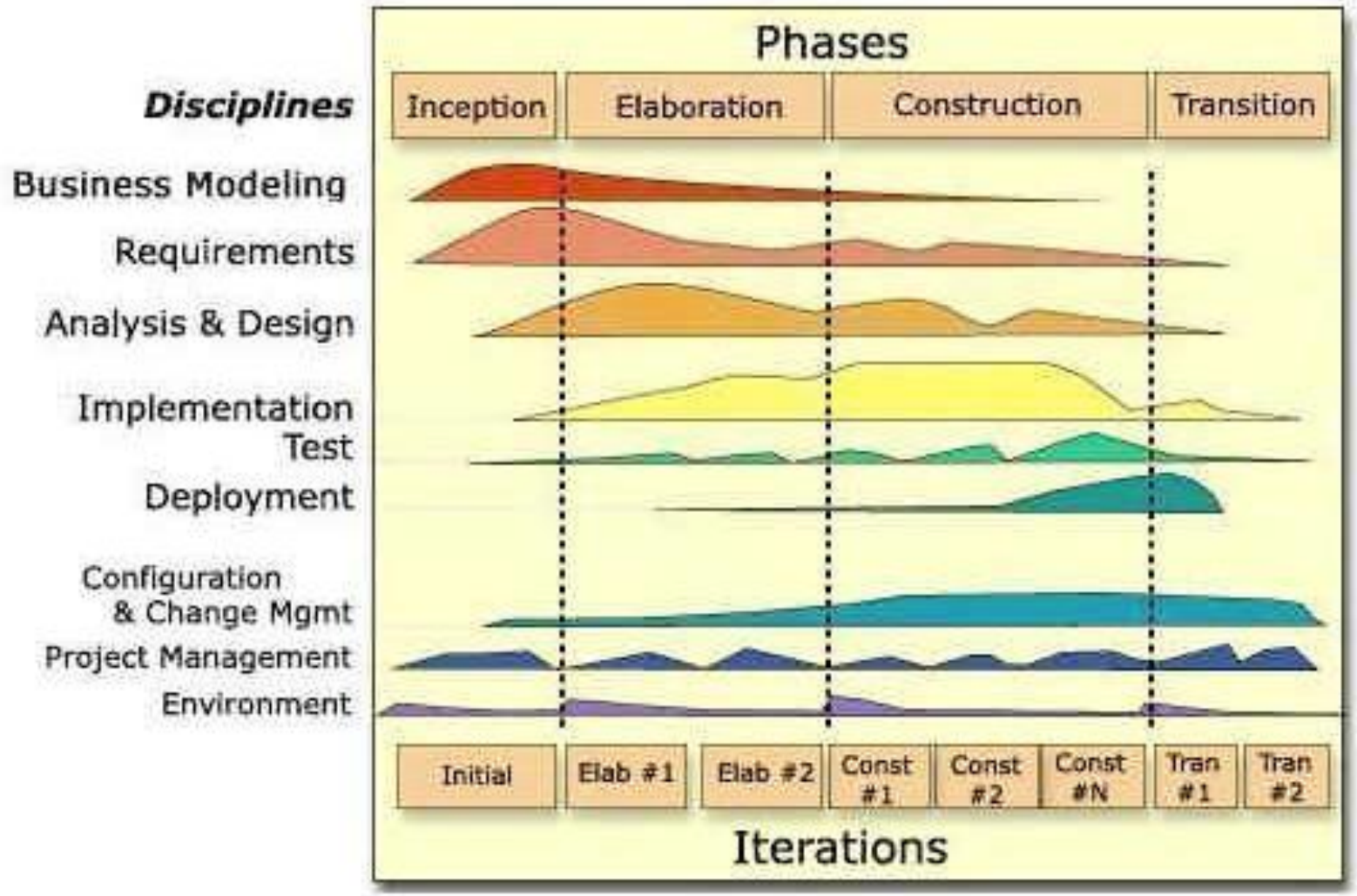
***Configuration and Change Management*** :manage access to project work products

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

***Environment***: ensure that process, guidance and tools are available



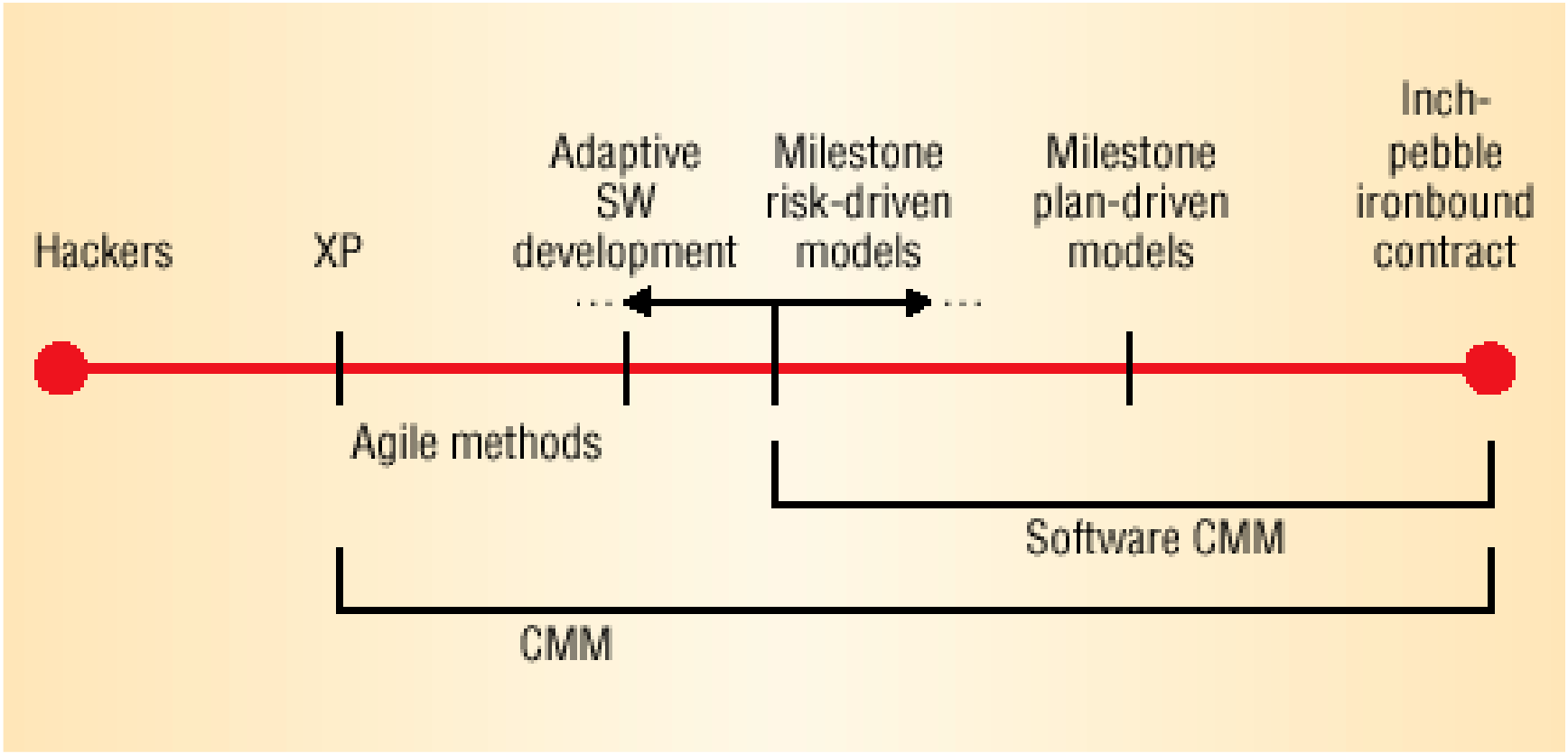
# RUP Phases, Iterations and Disciplines



# Agenda

- Software development method comparison
- Motivations for agile methods
  - 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

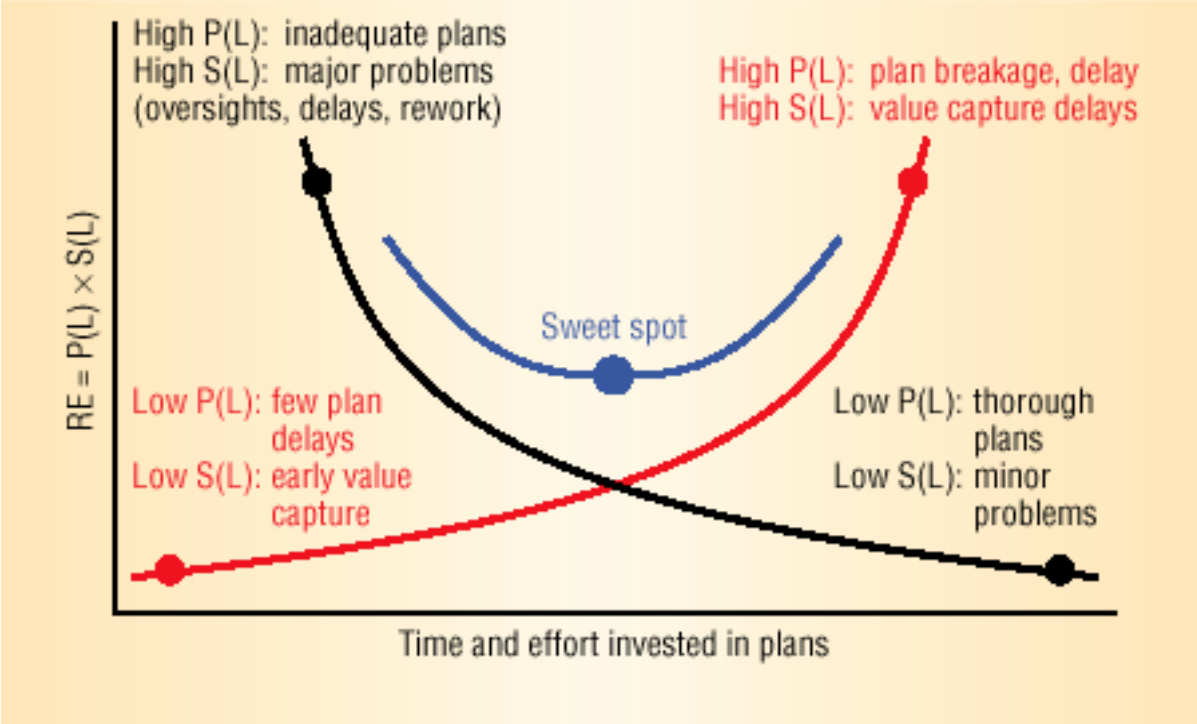



***Figure 2. Risk exposure (RE) profile. This planning detail for a sample e-services company shows the probability of loss P(L) and size of loss S(L) for several significant factors.***

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

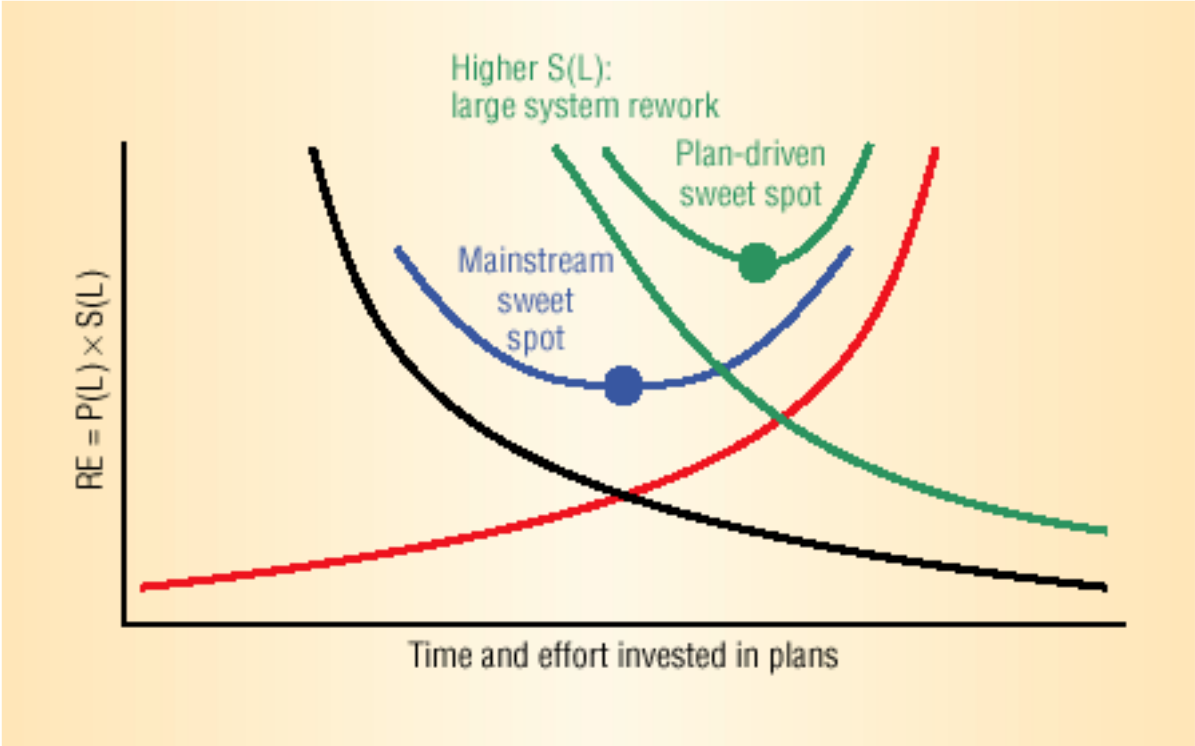



***Figure 4. Comparative RE profile for a plan-driven home-ground company that produces large, safety-critical systems.***

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

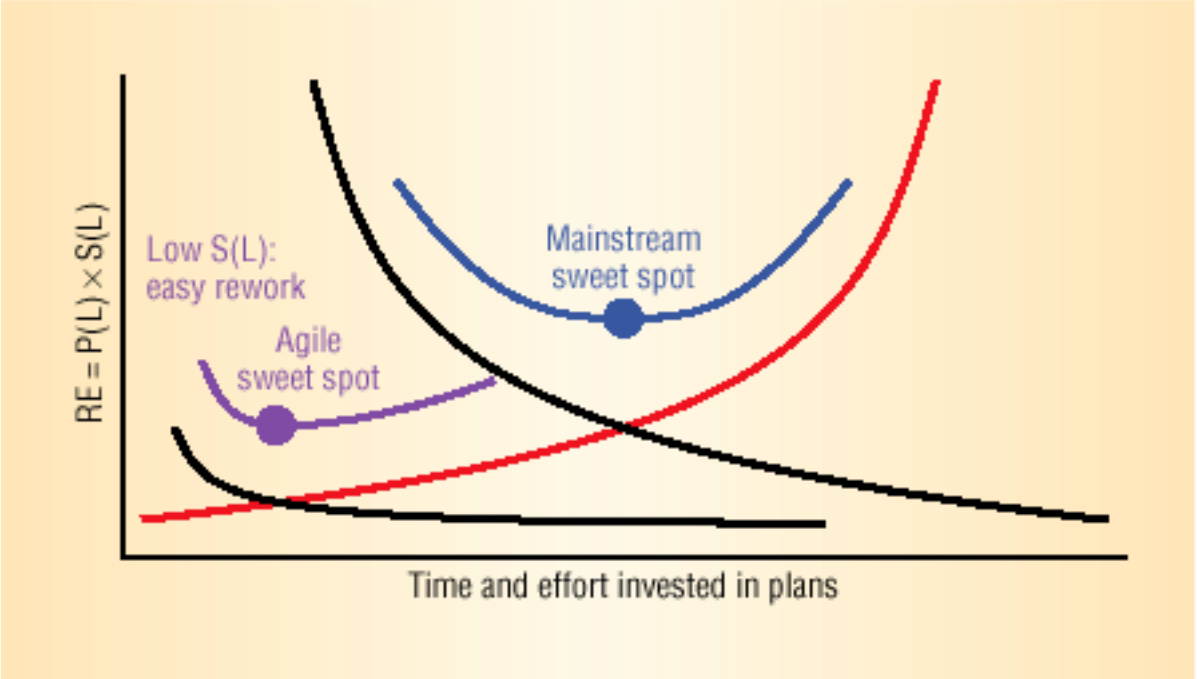



***Figure 3. Comparative RE profile for an agile home-ground company with a small installed base and less need for high assurance.***

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

# Agile Manifesto (2001)

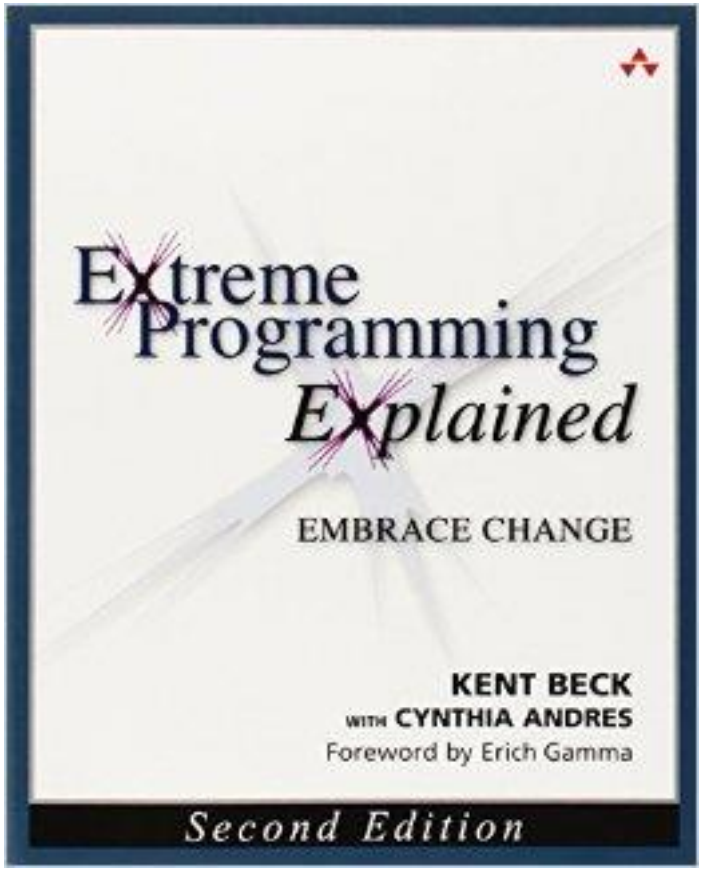

12 Principles behind the Agile Manifesto
http://agilemanifesto.org/

# Agenda

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

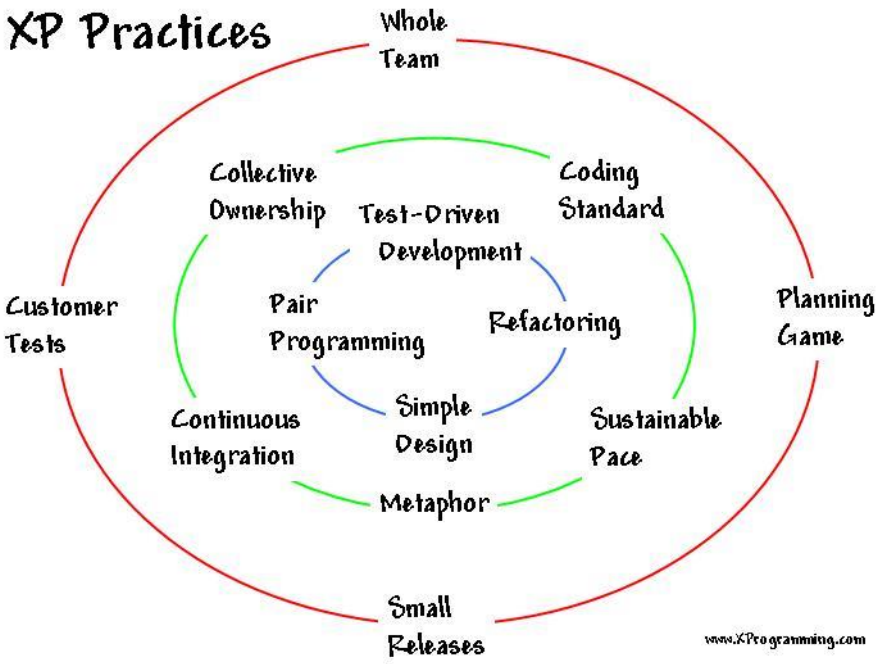
# Extreme Programming (XP)

- 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

1. The Planning Game
2. Small releases
3. Metaphor
4. Simple design
5. Testing
6. Refactoring
7. Pair programming
8. Collective ownership
9. Continuous integration
10. Sustainable pace
11. Whole team
12. Coding standards



# 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.

Business people 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. 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

# 3. 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

*E.g. "this program works like a hive of bees, going out for pollen and bringing it back to the hive" as a description for an agent-based information retrieval system.*

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

# 4. 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

# 5. 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

# 6. 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 of the tests.

The advantages of Refactoring are:

✓ It become easier to make the next changes

✓ Increases the developer knowledge of the system

# 7. 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

# 8. 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.

# 9. 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.

# 10. 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.

# 11. 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

# 12. 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

# Workspace

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


STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

**stevens.edu**

Thank You