CS350 Lecture
Software Dev. Life Cycle

CS350 Fall 2018

Doo-Hwan Bae

bae@se.kaist.ac.kr

Contents

Programming Paradigm (1/2)

- Paradigm: refer to a category of entities that share a common characteristic
- Programming paradigm
  - A model for a class of Programming Languages(PL) that share a set of common characteristics
  - A PL is a system for specifying a computation(data and logic) to communicate with a computer, that is, to execute the computation.
  - Each PL is defined with its syntax and semantics.

Programming Paradigm (2/2)

- Imperative paradigm
  - Fortran, C, Pascal, COBOL,
- Object-oriented paradigm
  - Simula67, Smalltalk, C++, Java,…
- Logic paradigm
  - Prolog,
- Functional paradigm
  - Lisp, ML, Haskell
- Hybrid paradigm: support multi-paradigms
  - …..??

Programming Language Popularity

- Top 10 by Redmonk , IEEE Spectrum, PYPL

	Redmonk	IEEE Spectrum	PYPL
1	JavaScript	Python	Java
2	Java	C	Python
3	Python	Java	PHP
4	PHP	C++	C#
5	C#	C#	JavaScript
6	C++	R	C++
7	CSS	JavaScript	C
8	Ruby	PHP	R
9	C	Go	Object-C
10	Objective-C	Swift	Swift

Software Development Paradigm

- Closely related to programming paradigms
  - Not just coding, but modeling (analysis&design)
  - Also, software structure
- What is the basic element of software? (building block)
  - Function
  - Procedure
  - Object/class
  - Component
  - (Web) Service
  - (Software) System

Comparison of Basic Elements (Building Blocks)

- With respect to SE concerns, such as
  - granularity(size), software reuse, standardization, quality, etc

SW Development Life Cycle

- Build-and-Fix
- Waterfall
- Prototyping
- Incremental & Iterative
- Transformational
- Spiral
- V model
- Agile

What is a Software Process Model?(1/2)

- Boehm(1988): its goal

  "determines the order of stages involved in S/W development and evolution, and to establish the transition criteria for progressing one stage to the next. These include completion criteria for the current stage plus choice criteria and entrance for the next stage. Thus a process model addresses the following S/W project questions:

  - What shall we do next?
  - How long shall we continue to do it?

- Other names: Software production process, Software (dev.) life cycle model, etc.

What is a Software Process Model?(2/2)

- Process models have a twofold effect

  - They provide guidance to software engineers on the order in which the various technical activities should be carried out within a project.

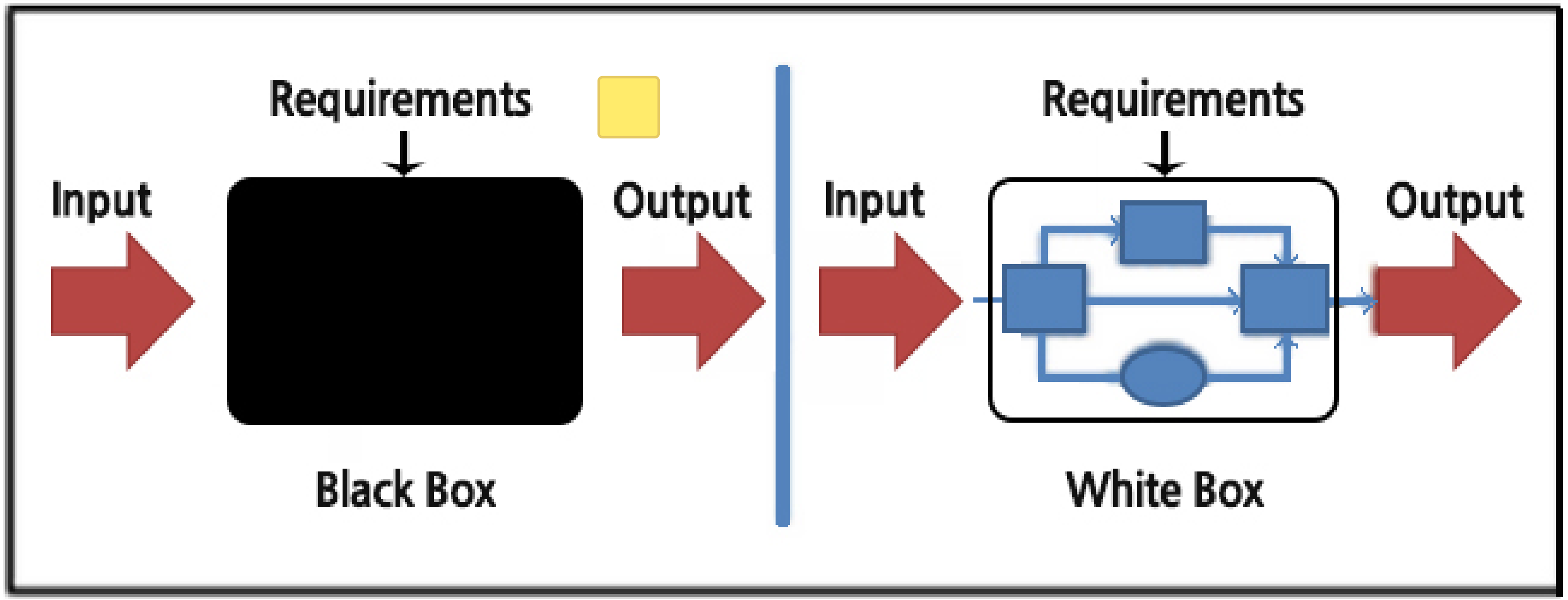
  - They afford a framework for managing development and maintenance, in that they enable us to estimate resources, define intermediate milestones, monitor progress, etc.

Why are SW Process Models Important? (1/3)

- “Processes are important because industry cares about their intrinsic qualities, such as uniformity of performance across different projects and productivity, with the aim of improving time to market and reducing production costs.”
- “But processes are also important … that they have a decisive influence on the _______ of products; that is, by controlling processes, we can achieve better control of the required _____ of products.

Why are SW Process Models Important? (2/3)

- The process as a black box
- The process as a white box



Why are SW Process Models Important? (3/3)

- What are the differences? Which one do you prefer?
  - As a developer?
  - Or as a project manager?

- Process viewed as a black box

  Q: what are the problems in the black box process?

  ➔ "What You See Is What You Get" (WYSIWYG)

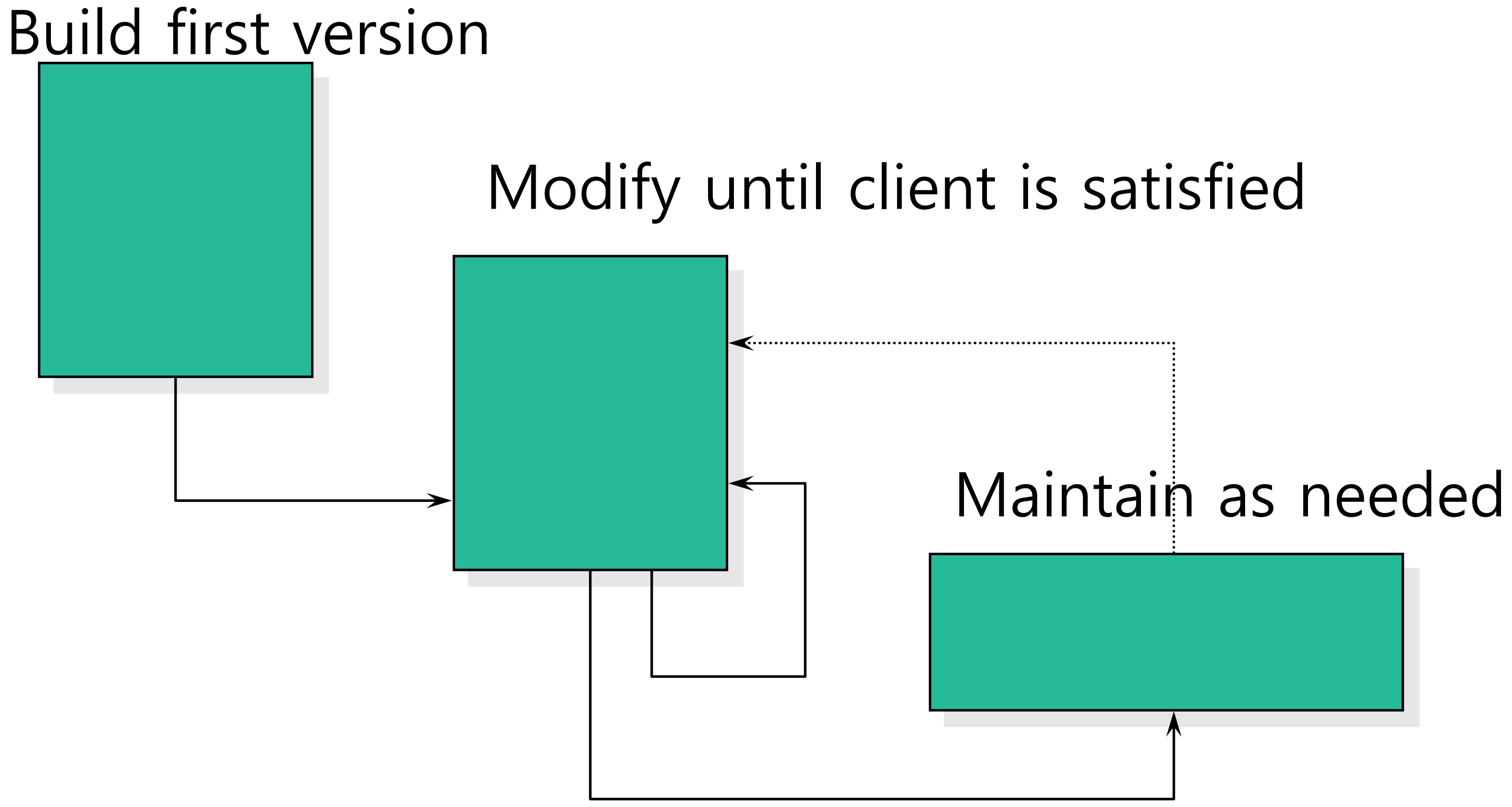
  ➔ Make invisible software development 'Visible'!

- Process quality will lead to ______ quality!!

Software Development Life Cycle Models

- Build-and-Fix
- Waterfall
- Prototyping
- Incremental & Iterative
- Transformational
- Spiral
- V model
- Agile
- Others

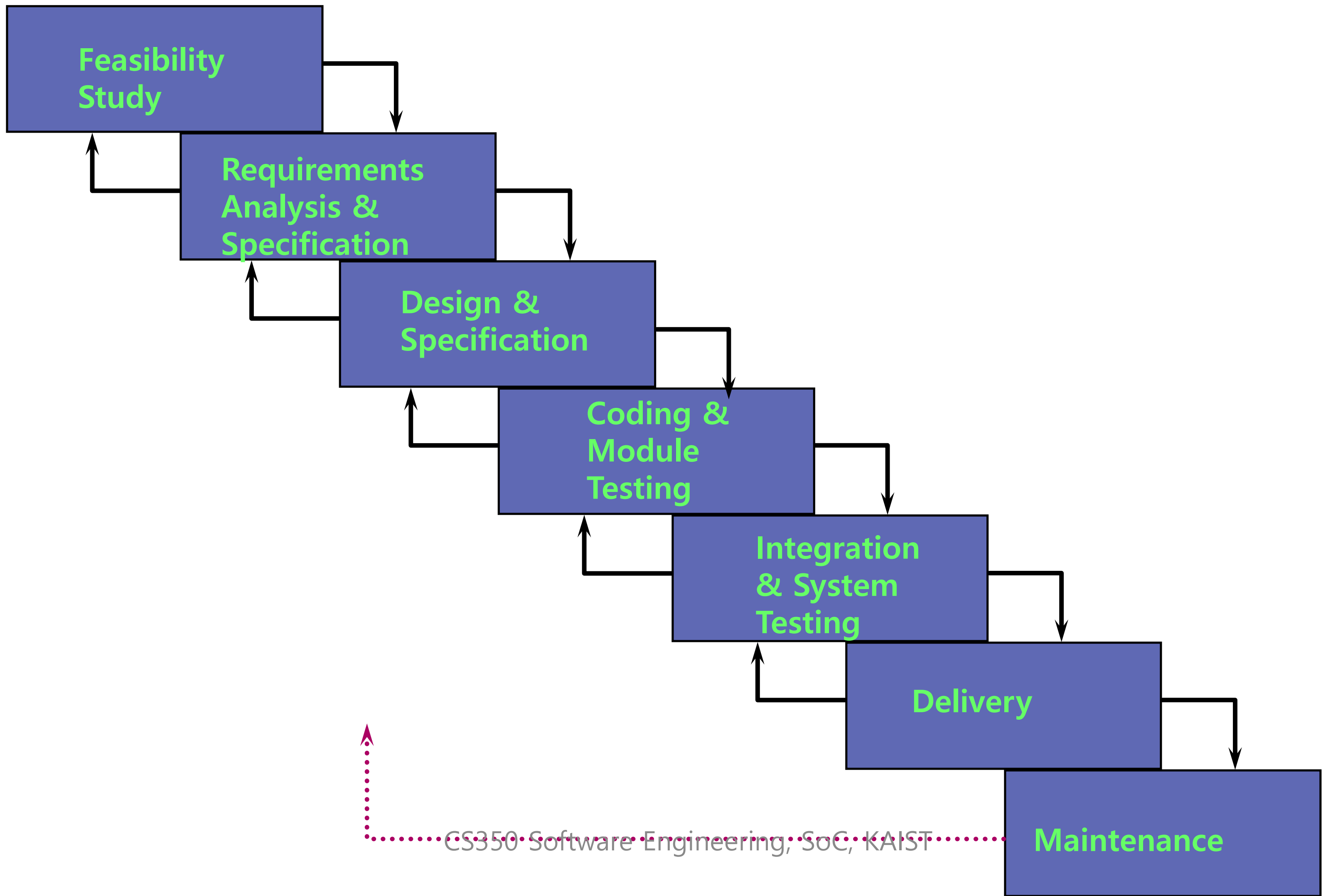
Build-and-Fix Model (Code-and-Fix Model) (1/2)



Build and Fix Model(2/2)

- Mainly a single-person task.
- Constructed without specification.
- Unsatisfactory for products of reasonable size.
- Not suitable for today's environments where
  - More stringent reliability requirements
  - Group activity.

Waterfall Model(1/11)



Waterfall Model(2/11)

- Popularized in 1970's.
- Standard industrial practices.
- Document-oriented
- The output of one phase constitutes the input to next.
- Exists many variants.

Waterfall Model: Documents (3/11) [Sommerville]

Activity

- Requirements analysis
- Requirements definition
- System specification
- Architectural design
- Interface design
- Detailed design
- Coding
- Unit testing
- Module testing
- Integration testing
- System testing
- Acceptance testing

Output document

- Feasibility study
- Outline requirements
- Requirements specification
- Functional specification
- Acceptance test specification
- Draft user manual
- Design architecture spec
- System test specification
- Interface specification
- Integration test specification
- Design specification
- Unit test specification
- Program code
- Unit test result report
- Module test result report
- Integration test result report
- Final user manual
- System test result report
- Final system test result report

Feasibility Study(4/11)

- Evaluates the costs and benefits of the proposed solution.
- Analyze the problem, at least at the global level.
- Simulation of the future development process.
- Documents produced:
  - A definition of the problem.
  - Alternative solutions and their expected benefits.
  - Required resources, costs, and delivery date in each.

Requirements Analysis and Specification(5/11)

- Identify the qualities required for the application.
  - Functional and nonfunctional
- Must state what to do, not how to do.
- Used by both customers and designers
- Separate functional requirements into three views:
  - Q: What are they?

Design and Specification (6/11)

- Propose a solution to the problem.
- Decompose the system into modules.
  - Specify the relationships among modules.
  - Design each module.
- Can be divided into
  - High-level(preliminary, architectural)
  - Low-level(detailed)
  - User interface

Coding and Testing(7/11)

- Ideally transform of design into code
- Testing
    - Unit (module) testing
    - Integration testing
    - System testing
    - Acceptance testing
    - Etc….

Delivery and Maintenance(8/11)

- Types of maintenance
  - Corrective: 20 %
  - Adaptive: 20 %
  - Perfective: 60%
- Requirements analysis is a serious source of problems.
- Many errors are not removed until after the system is delivered.
- It is difficult to incorporate changes in the product.

Other Activities(9/11)

- Documentation
  - Document-driven
- Verification
  - Monitor the quality of the application.
  - Perform at the end of each phase.
  - Methods
    - Review
    - Walk-through
    - Inspection
- Management
  - Tailoring the process.
  - Defining policies.
  - Dealing all resources affecting the process.

Evaluation of the Waterfall Model(10/11)

- Linear
  - Development may proceed linearly from analysis to coding.
- Rigid
  - The results of each phase are frozen before proceeding to the next.
- Monolithic
  - All planning is oriented toward a single delivery date.

Evaluation of the Waterfall Model(11/11)

- Contribution
  - Enforced disciplined, planned and manageable approach.
  - Implementing the product should be postponed until after the objectives of doing so are well understood.
- Problems
  - Difficult to estimate resources accurately.
  - Verification of req. spec. by customer is not effective.
  - The user often does not know the exact requirements.
  - Does not stress the need for anticipating changes.
  - Leads to a somewhat bureaucratic style of work.

Prototyping(1/2)

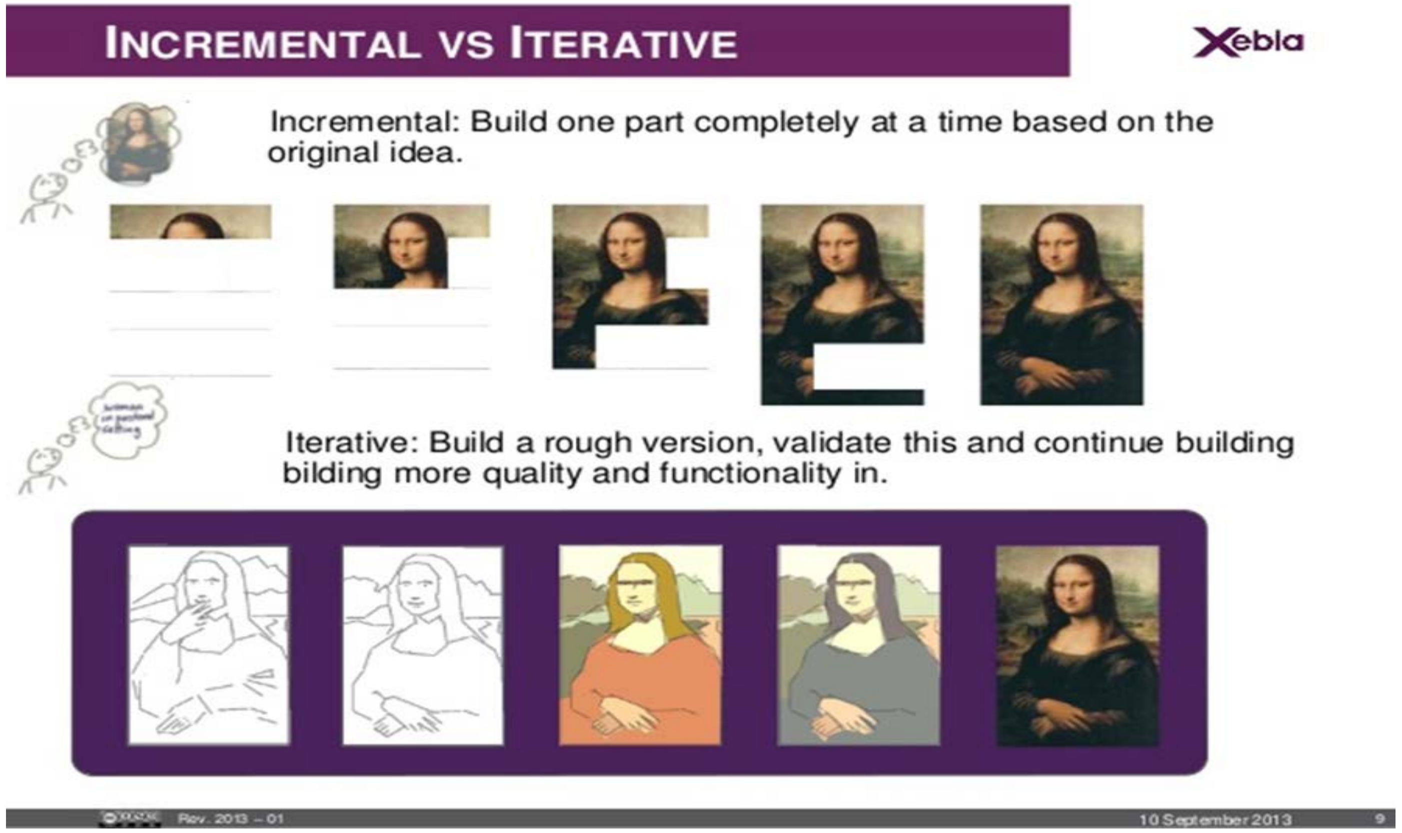
- Prototyping
  - Throwaway prototyping
  - Incremental(evolutionary) prototyping
- Throwaway prototyping
  - "Do it twice"
  - Used temporarily, until it provides enough feedback.
  - Working model, functionally equivalent to a subset of the product.
  - Can be used in conjunction with other models.

Prototyping(2/2)

- Benefits
  - Helps to reduce the cost and time.
  - Improves communication.
  - Helps to detect errors early.
  - Lets the developers have greater control over the problem.
- Problems
  - Does not eliminate the time gap, between the definitions of requirements and final delivery of the application
  - Does not stress the need for anticipating changes.
  - Has a bad reputation, "a quick and dirty solution."
  - Suffers from bad documentation.
  - Not a stand alone model

Incremental & Iterative (1/4)

- Incremental
- Iterative



INCREMENTAL VS ITERATIVE

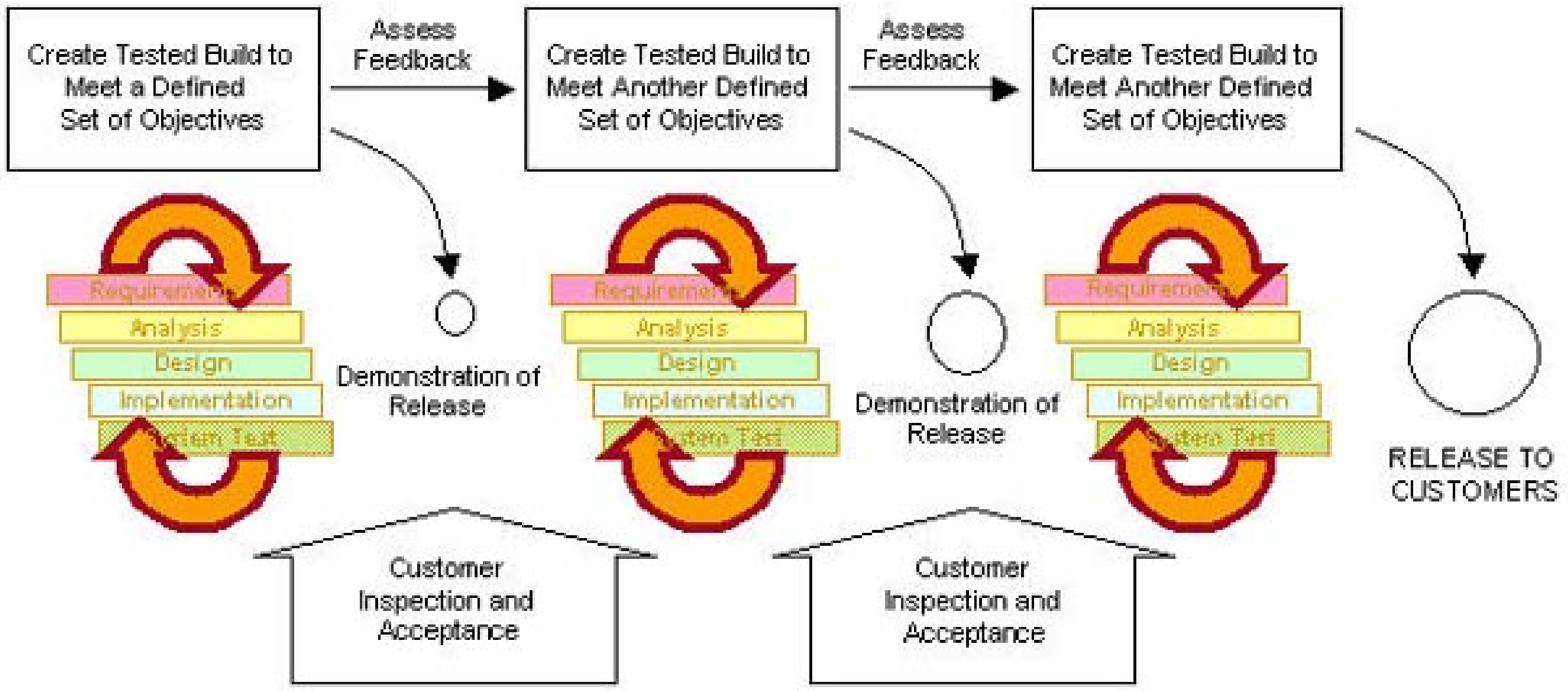
Incremental: Build one part completely at a time based on the original idea.

Iterative: Build a rough version, validate this and continue building bilding more quality and functionality in.

Rev. 2013 – 01 10 September 2013 9

Incremental and iterative (2/4)

- Phased delivery


Create Tested Build to Meet a Defined Set of Objectives
Assess Feedback
Create Tested Build to Meet Another Defined Set of Objectives
Assess Feedback
Create Tested Build to Meet Another Defined Set of Objectives
Requirements
Analysis
Design
Implementation
System Test
Demonstration of Release
Demonstration of Release
RELEASE TO CUSTOMERS
Customer Inspection and Acceptance
Customer Inspection and Acceptance

Incremental and Iterative(3/4)

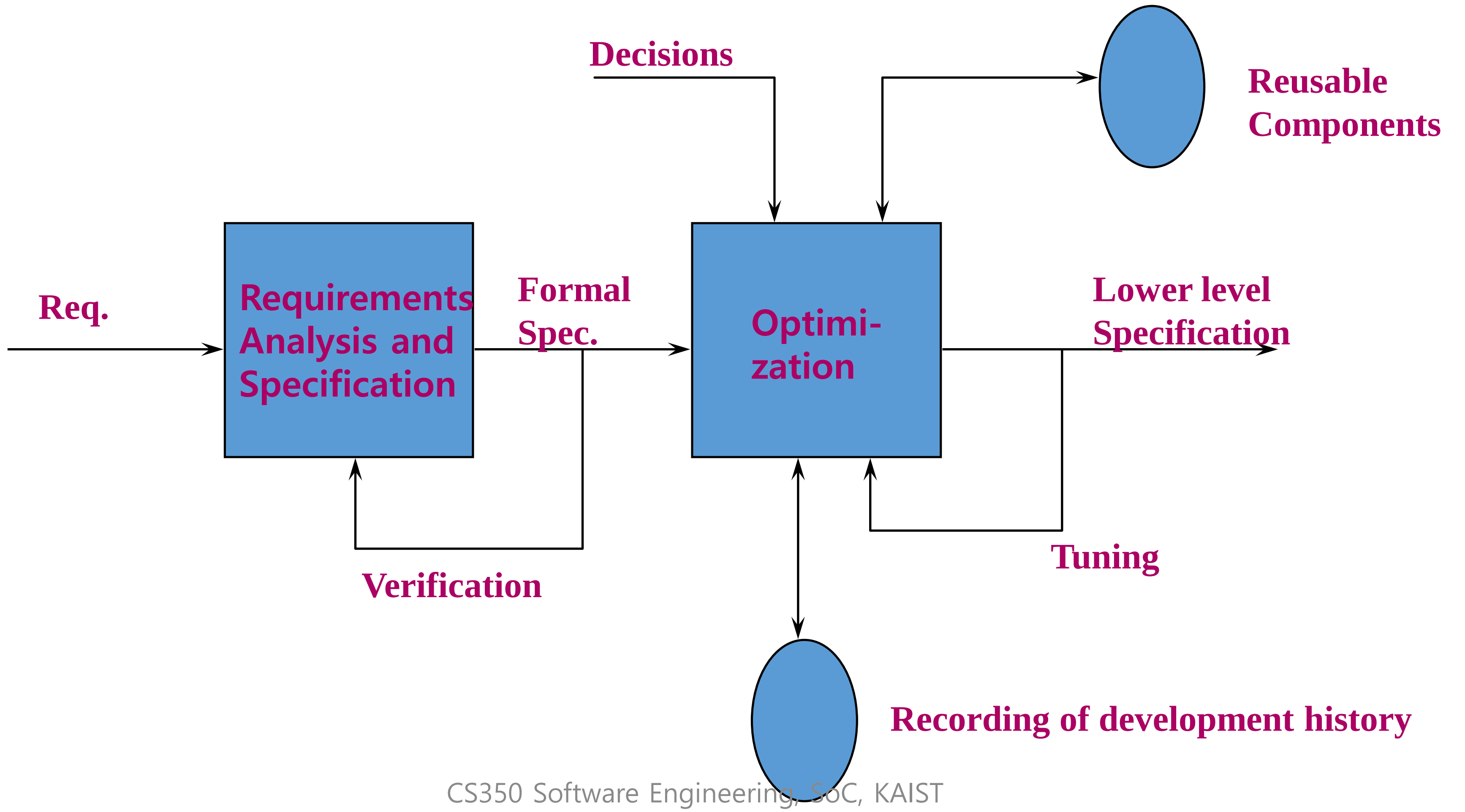
- Stepwise development
- Must retain the discipline introduced by the waterfall model at each build.
- May be extended to all stages of the life cycle.

Incremental and Iterative (4/4)

- Benefits
  - Provide the user with time to adjust to the new product.
  - Easy to accommodate changes.
  - Phased delivery does not require a large capital outlay.
- Problems
  - Each additional build has to be incorporated into the existing structure.
  - May require more careful design. (could be a benefit.)
  - Can easily degenerate into the build&fix model.
- Discuss possible incremental implementation strategies.

  - top-down, bottom-up, hybrid, scenario-based, big-bang, …

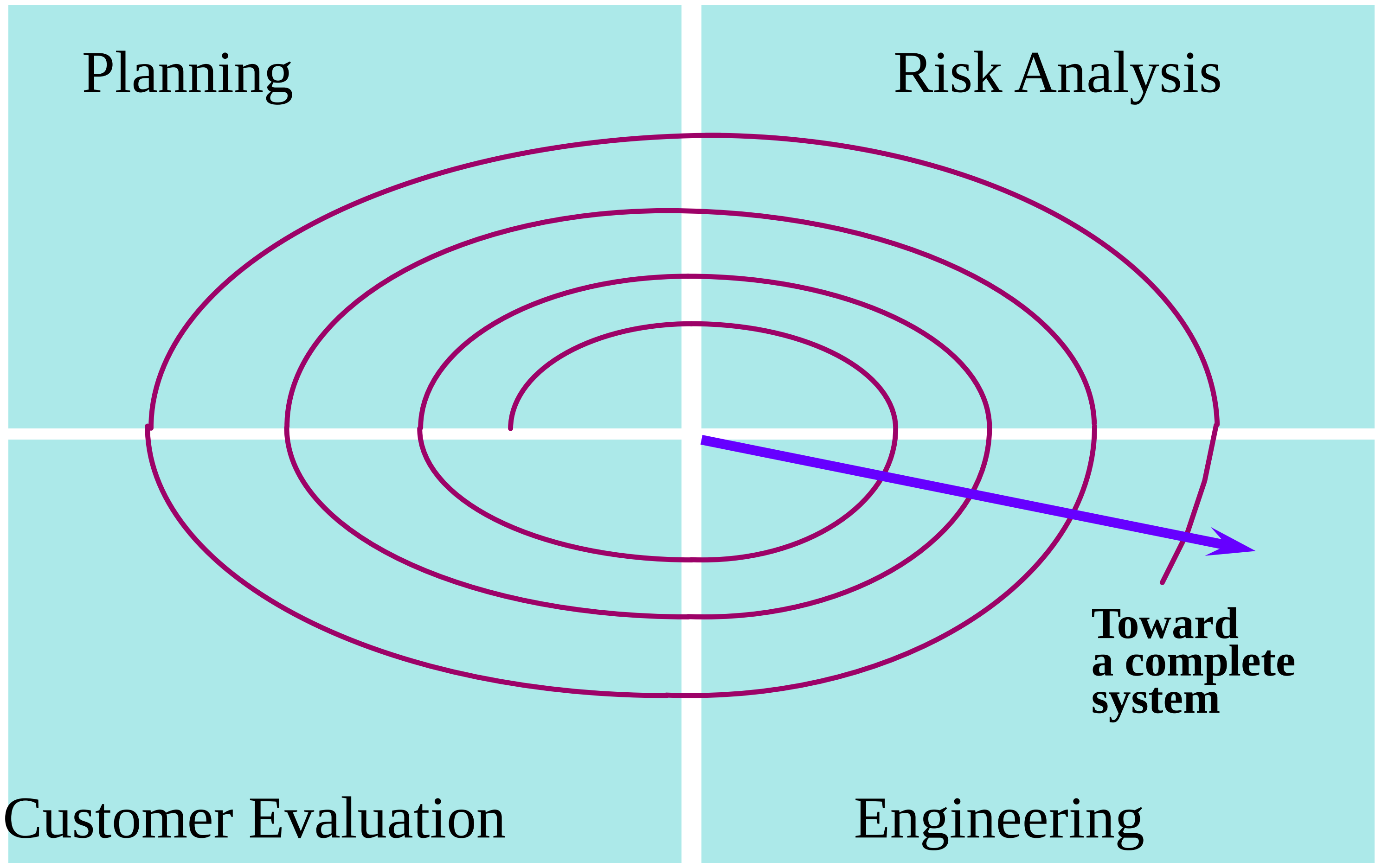
Transformation Model(1/2)


Decisions
Reusable Components
Req.
Requirements Analysis and Specification
Formal Spec.
Optimi-zation
Lower level Specification
Verification
Tuning
Recording of development history

Transformational Model(2/2)

- Based on formal Specification
- Software development viewed as a sequence of steps that gradually transform a spec. into an implementation.
- Q: What are industrial examples of transformational approach?

Spiral Model(1/3)



Spiral Model(2/3)

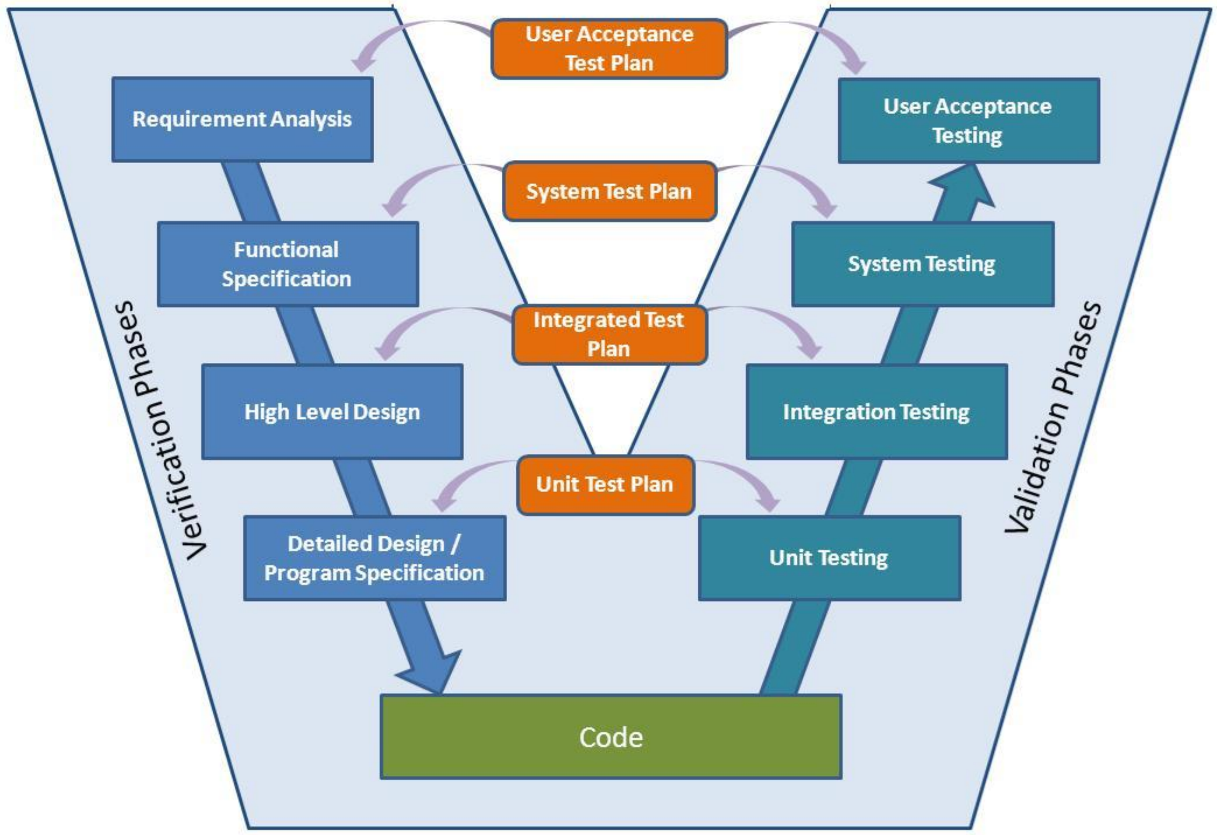
- Developed by Barry Boehm
- To provide a framework for designing the software production process, guided by the risk levels in the project at hand
- Viewed as a meta-model

Spiral Model(3/3)

- Benefits
  - The emphasis on automation and constrains supports the reuse of existing s/w, and the incorporation of s/w quality as a specific objective
  - No distinction between maintenance and development (another cycle of the spiral)
- Problems
  - Restriction on the range of its applicability (intended exclusively for internal development of large scale software)
  - may cost too much, due to the risk analysis at each spiral
- I expect you to be able to select a right model for your project!

V model

- Testing emphasized
- Youtube: “SDLC and STLC”



Agile Software Development

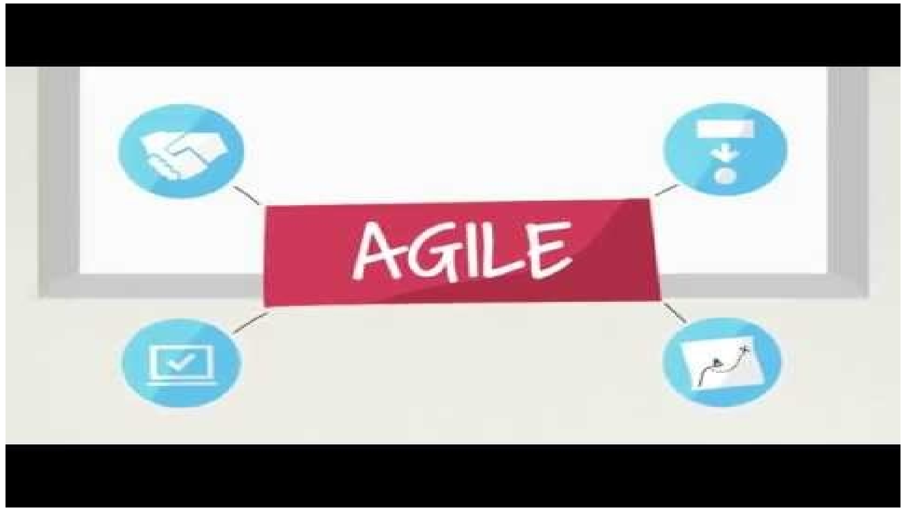
- February 2001, seventeen software developers met at the Snowbird resort in Utah to discuss lightweight development methods, among others Jeff Sutherland, Ken Schwaber, and Alistair Cockburn. Together the seventeen published the ***Manifesto for Agile Software Development*.**
  - **Individuals and Interactions** over processes and tools
  - **Working Software** over comprehensive documentation
  - **Customer Collaboration** over contract negotiation
  - **Responding to Change** over following a plan

Agile Principles

- Customer satisfaction by early and continuous delivery of valuable software
- Welcome changing requirements, even in late development
- Working software is delivered frequently (weeks rather than months)
- Close, daily cooperation between business people and developers
- Projects are built around motivated individuals, who should be trusted
- Face-to-face conversation is the best form of communication (co-location)
- Working software is the principal measure of progress
- Sustainable development, able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicity—the art of maximizing the amount of work not done—is essential
- Best architectures, requirements, and designs emerge from self-organizing teams
- Regularly, the team reflects on how to become more effective, and adjusts according

Agile: Manifesto, An Introduction

- Youtube: Agile: an introduction, manifesto



Agile Methods

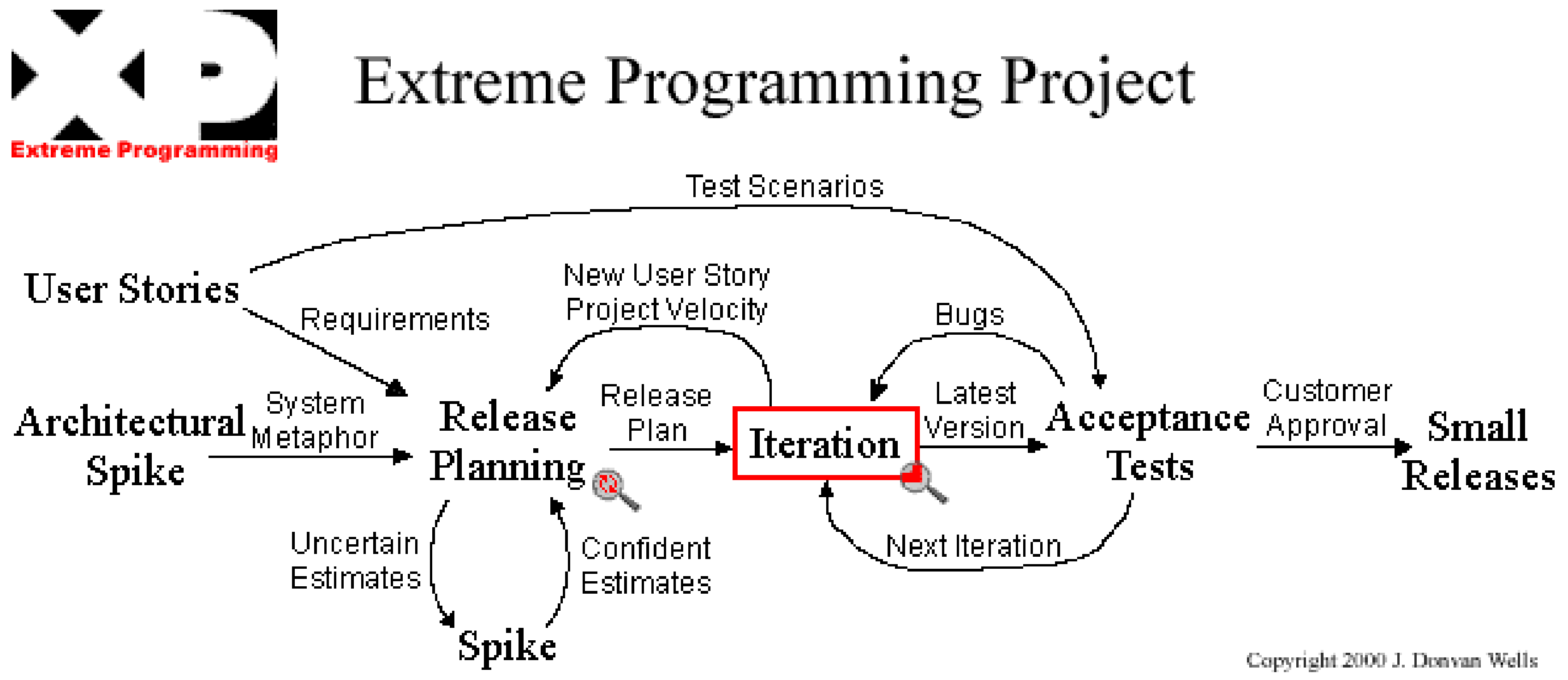
- Extreme Programming(XP)
- Dynamic Systems Development Method
- Lean
- Kanban
- Scrum
- Scrumban
- ….

XP(Extreme Programming)

- Planning Game
- Small Releases
- Customer Acceptance Tests
- Simple Design
- Pair Programming
- Test-Driven Development
- Refactoring
- Continuous Integration
- Collective Code Ownership
- Coding Standards
- Metaphor
- Sustainable Pace

XP

- Frequent releases for improving quality and customer involvement



DSDM

- Fitness for business purpose
- Focusing on the useful 80% of the system that can be delivered in 20% of the time.
- MoSCow rules
    - *M:Must have requirements*
    - *S: Should have if at all possible*
    - *C: Could have but not critical*
    - *W: Won 't have this time, but potentially later*

Lean

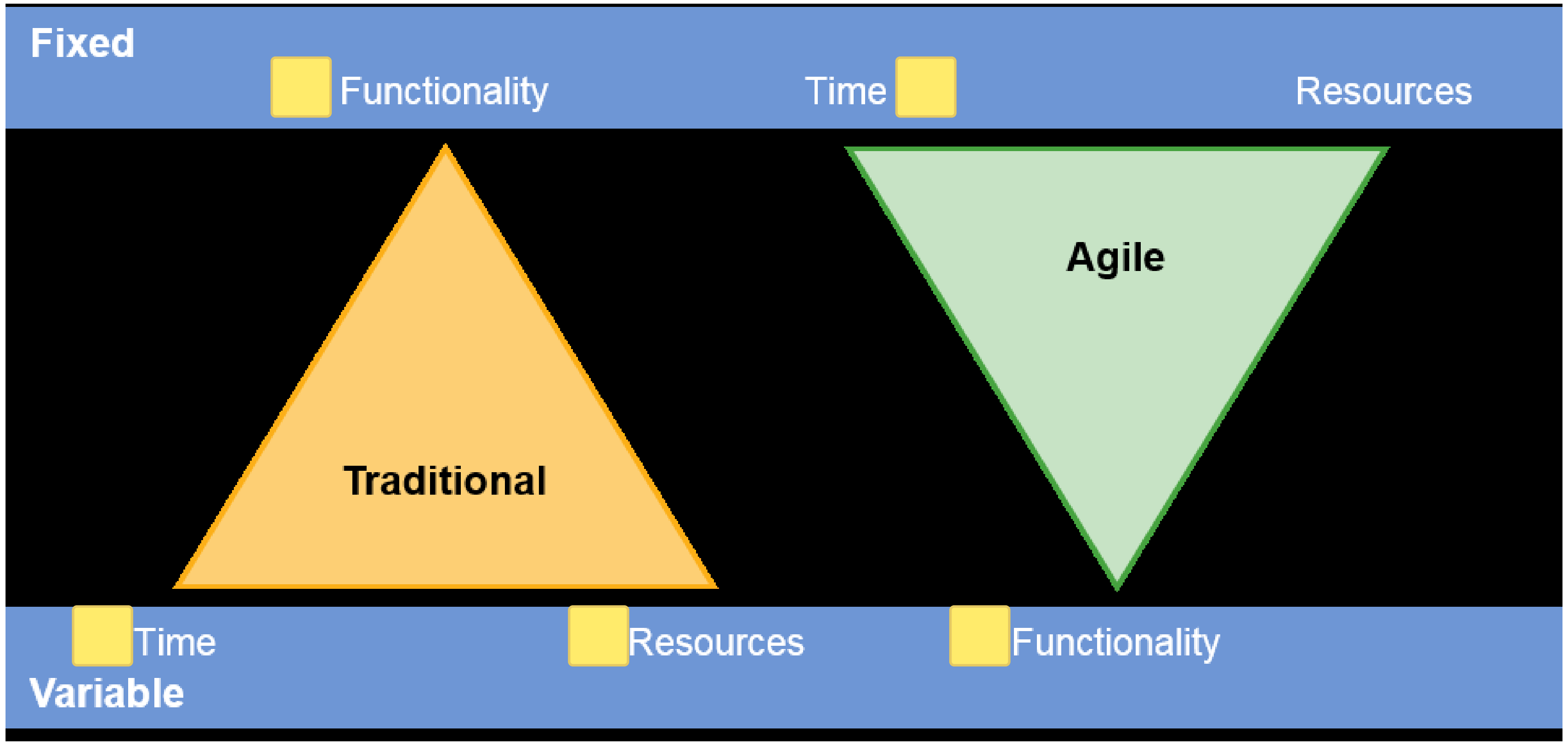
- Originated from Toyota Auto manufacturing practices
- Main principles
  - Eliminating Waste
  - Amplifying Learning
  - Deciding as Late as Possible
  - Delivering as Fast as Possible
  - Empowering the Team
  - Building Integrity In
  - Seeing the Whole

Kanban

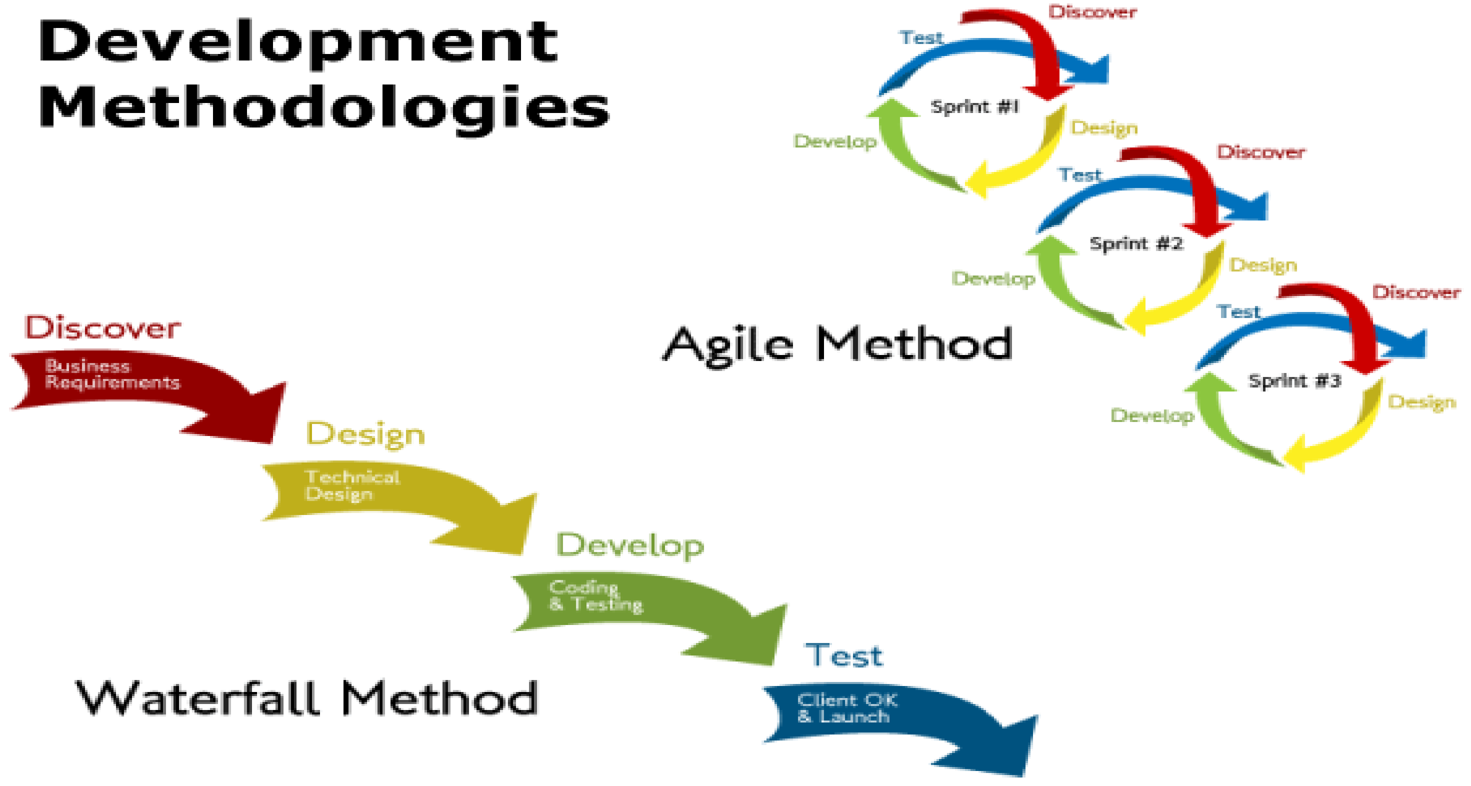
- Used by organizations by the creation of products with an emphasis on continual delivery while not overburdening the development team
- Based on 3 basic principles
  - Visualize what you do today
  - Limit the amount of work in progress
  - Enhance flow

Comparison of Waterfall and Agile Dev.(1/3)
(source: Gartner 2016)

- With Respect to Functionality, Time, and Resources

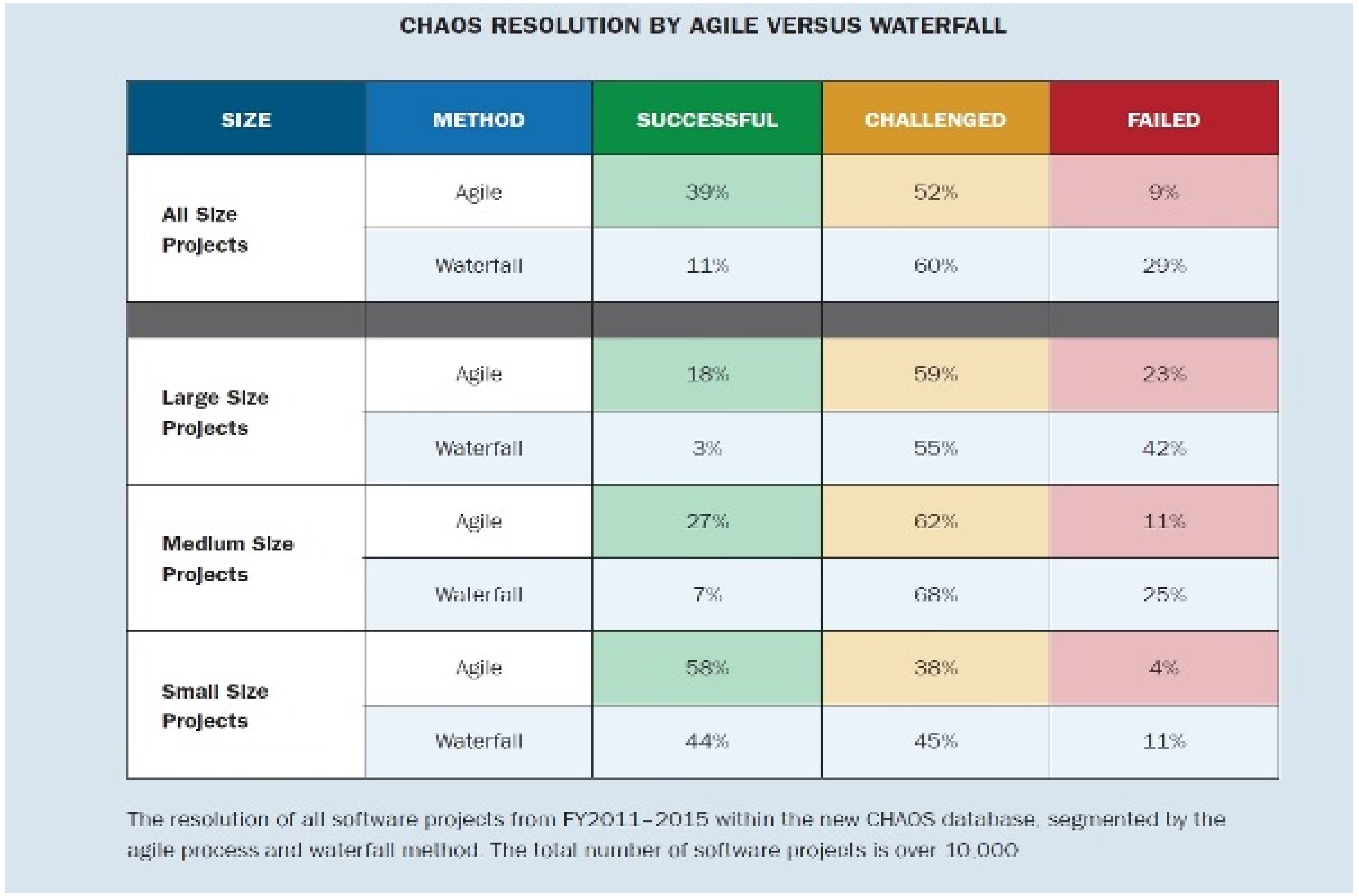


Comparison of Waterfall and Agile Dev.(2/3)



Comparison of Waterfall and Agile Dev.(3/3)
(Source: Standish Group 2015CHAOS Report)

- Success Rate

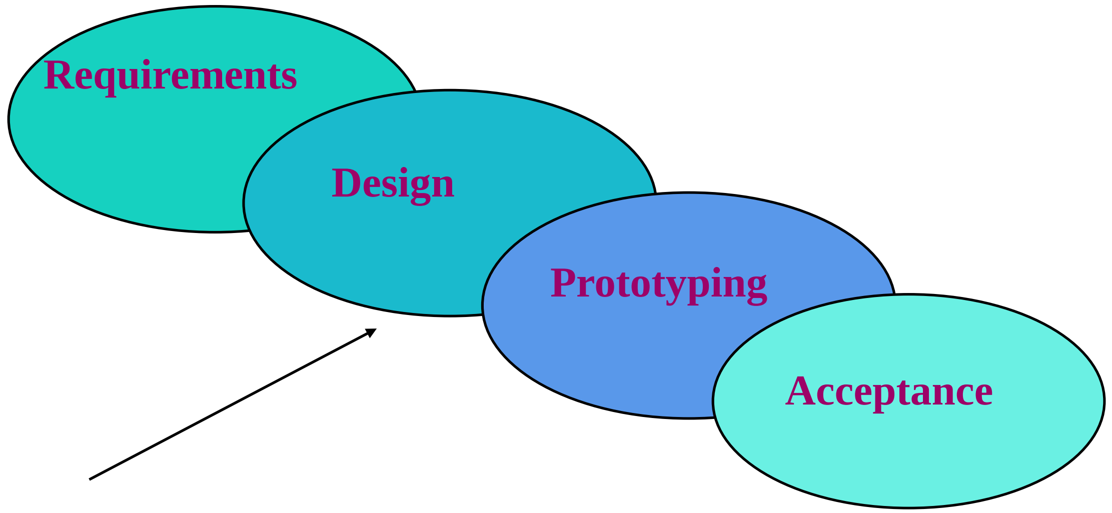


CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011-2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000

Others



- Sashimi's team approach
  - Similar to waterfall model except neighboring phases are overlapped.
- Two man handcuffing

So far,

- Build and fix
  - SW Crisis observed; Engineering practice needed
- Waterfall model
  - Successful engineering principles for hardware applied
  - Due to the differences between SW and HW, it did not work well!
- Many attempts to resolve problems found
  - Prototyping, iterative&incremental, agile,..
- There is no absolute solution working for every project
- Need expertise to select the right model for your project!

New Paradigm Emerged?

- Do agile methods really solve the problems associated with traditional SW development models?
- What attempts are made in internet service companies such as Googles, Netflix, Amazon, etc?
- Are traditional SE approaches working for them?

DevOps

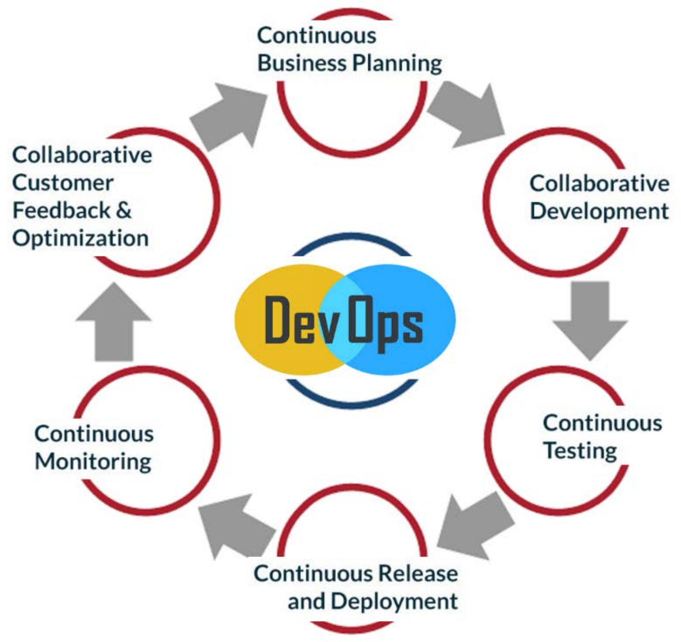
(source: Guest Editor's Introduction, IEEE Software May/June 2016)

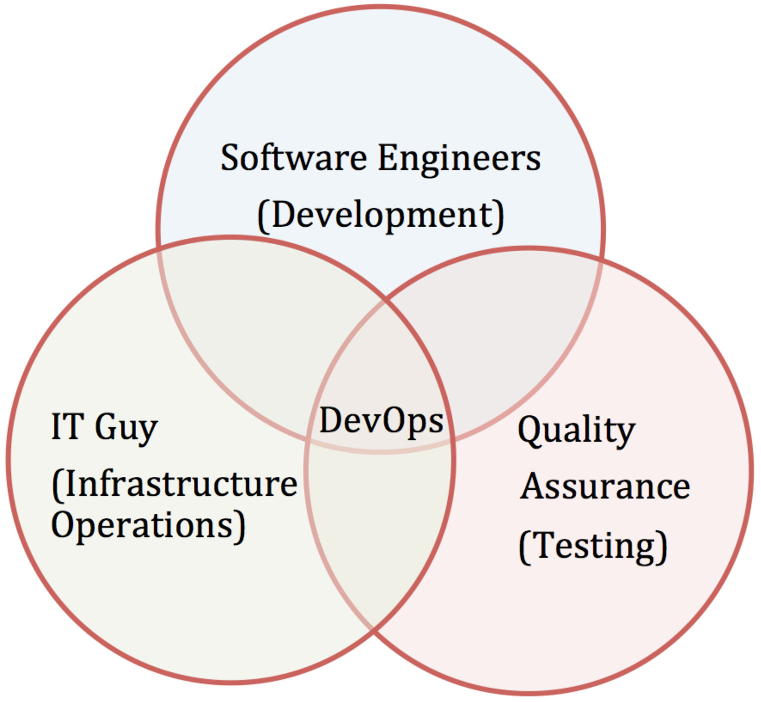
- A set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality
- Effects of DevOps
    - Developers verify a system's provenance upon initialization
    - Continuous development
    - Systems move through various environments on their way to production
    - Systems are monitored after deployment, and changes might be rolled back.
    - Rely heavily on tools, including container management, continuous integration, orchestration, monitoring, deployment, and testing

DevOps

- Microservices architectural style: becoming the standard for building continuously deployed systems
    - Restriction of a service-oriented architecture
    - Each service is small
    - All service developers understand they are working on the same overall system
    - Shares some characteristics of System of Systems

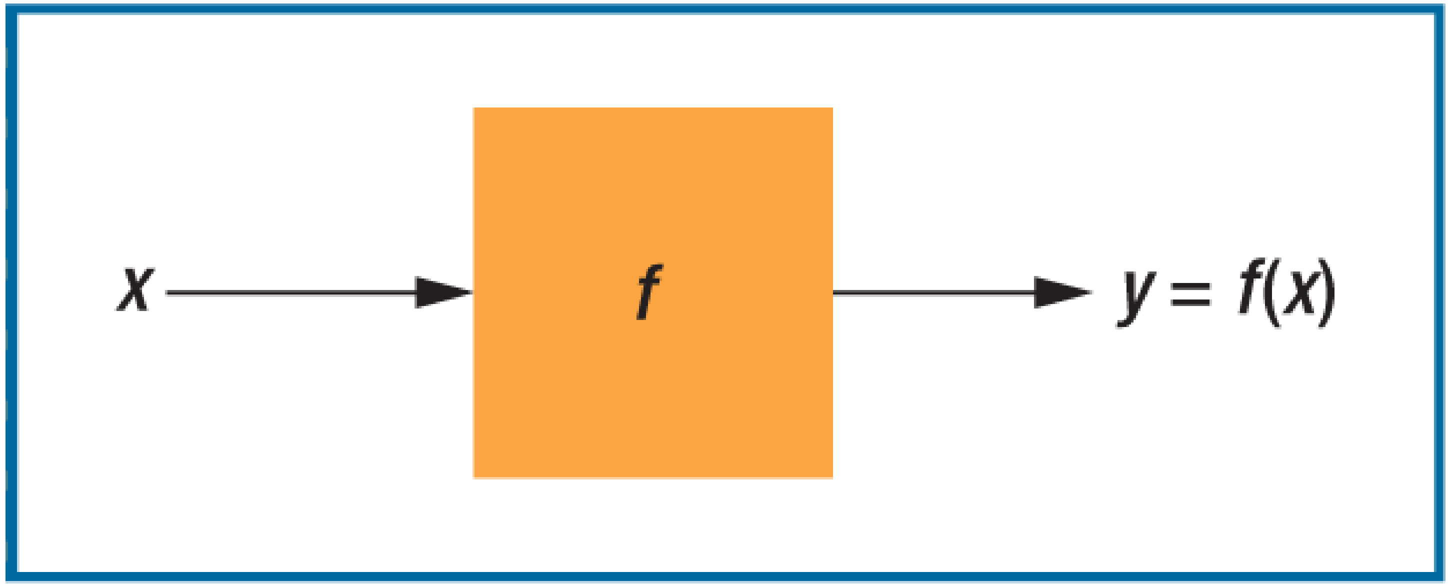
DevOps


Continuous Business Planning
Collaborative Development
Continuous Testing
Continuous Release and Deployment
Continuous Monitoring
Collaborative Customer Feedback & Optimization
DevOps


Software Engineers (Development)
IT Guy (Infrastructure Operations)
DevOps
Quality Assurance (Testing)

A Functional Perspective of a SW system (Source: Chaos Engineering)

- For a distributed system, such functional spec is incomplete



Chaos Engineering (source: Jan/Feb., IEEE Software)

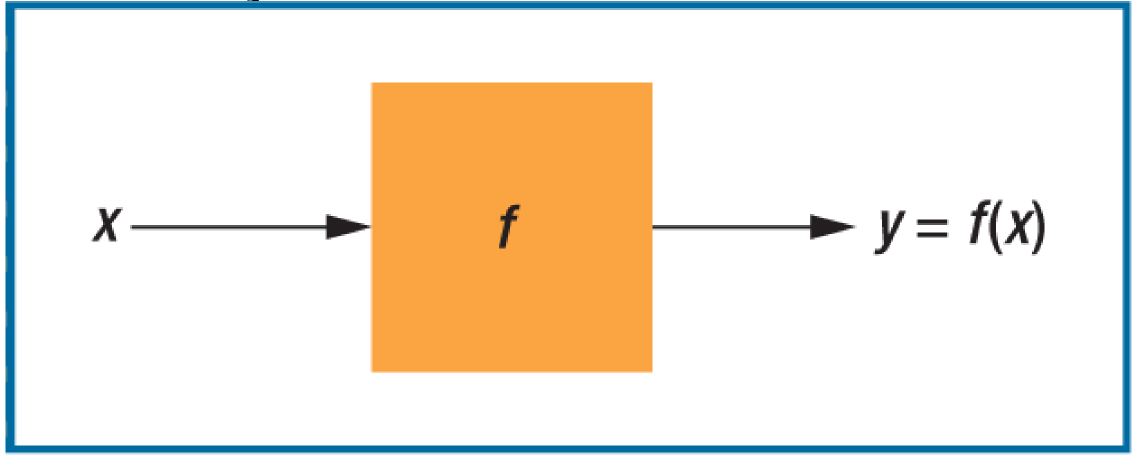
- Thirty years ago, Jim Gray noted, "A way to improve availability is to install proven hardware and software, and then leave it alone."
- Nowadays, does this policy still work?
- For companies providing services over internet, "leaving it alone" policy may not be an option at all.
  - What are such companies?
  - Is there any company that does not work over internet?

Chaos Engineering

- Chaos Engineering
  - Inject failures into the production system to improve reliability
    - Run failure injection testing exercises causing requests between Netflix services to fail and verify that the system degrades gracefully.
  - Netflix, Amazon, Google, Microsoft, Facebook do similar approach
  - A new discipline, emerging in industry
  - Involves experimenting on a distributed system to build confidence in its capability to withstand turbulent conditions
    - Hardware failure, an unexpected surge in client requests, runtime configuration error

Chaos Engineering

- Two premises
  - Engineers should view a collection of services running in production as a single system
  - We can better understand this system's behavior by injecting real-world inputs (for example, transient network failures) and observing what happens at the system boundary.
- In traditional approach, ➔



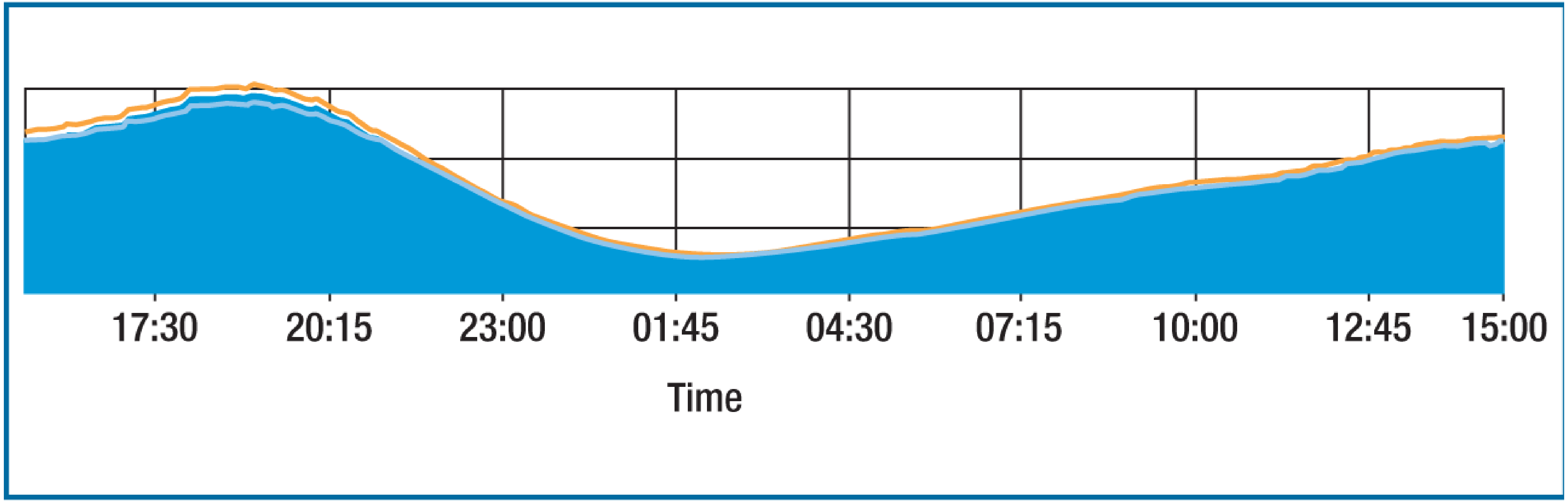
- In real-world,….system scale huge, not able to reason

Chaos Engineering

- Assumption on the system
    - The organization deployed the software into production some time ago and the system has active users
    - The engineers view the system as a single entity and observes its behavior by capturing metrics at its boundary.
    - Complex systems exhibit behaviors regular enough to be predicted
- The engineers cares about the typical behavior of metrics over time: system's steady-state behavior
    - The behavior can be changed by user interaction, engineer-initiated changes, and other events such as the transient failure of a third-party service
- Questions asked
    - Does everything seem to work properly?
    - Are users complaining?

A graph of SPS([streams] Start Per Second) over a 24 hour period at Netflix

- What we can learn from this graph?
  - Orange line shows the trend for the prior week. Y-axis, proprietary


17:30
20:15
23:00
01:45
04:30
07:15
10:00
12:45
15:00
Time