CS5351 Software Engineering 2024-2025 Semester A

Software Process Model

Dr W.K. Chan

Department of Computer Science

Email: **wkchan@cityu.edu.hk**

Website: **http://www.cs.cityu.edu.hk/~wkchan**

Motivation

- A group of ***strong and committed*** programmers (say G = {Programmer *A*, Programmer *B*}) can hack on a code project to deliver an application to their users.
  - *A* knows the code *P* and its responsibility extremely well
  - *B* knows the code *Q* and its responsibility extremely well
  - *A* can deliver the code *P*, and *B* can deliver the code *Q*
  - *A* meets *B*'s needs on *P*; *B* meets *A*'s needs on *Q*
  - *A* and *B* can check with users *U* to get and verify the exact user requirements whenever necessary
- A perfect case.

Motivation (in Reality)

- A group of **average, sometimes emotional,** programmers (say G = {Programmer *A*, Programmer *B*}) can hack on a code project to deliver an application to their users.
  - *A* knows the code *P* and its responsibility ***gradually***
  - *B* knows the code *Q* and its responsibility ***gradually***
  - *A* can deliver a ***buggy*** *P* with ***incomplete*** functionality, and *B* can deliver a ***buggy*** *Q* with ***incomplete*** functionality
  - *A* ***may not know*** *B*'s ***exact*** needs on *P*; *B* ***may not know*** *A*'s ***exact*** needs on *Q*. As *B* only knows *Q* gradually, *B*'s requirement on *P* changes over time.
  - *A* and *B* can check with users *U* to get and verify the exact user requirements ***sometimes***, but an ***old U resigns, and a replacement U comes***! Some *U*s are ***inexperience*** in their business domain.

Motivation (For Larger Project In Reality) [1/3]

Suppose in Month 1, the group G = {Programmer *A*, Programmer *B*}

- Month 2, G = {Programmer *B*}, *A* resigned.
- Month 3, G = {Programmer *A*1, Programmer *B*}, *A*1 replaces *A*.
- Month 4, G = {Programmer *A*1}, *B* resigned.
- Month 5, G = {Programmer *A*1, Programmer *B*1}, *B*1 replaces *B*.
- Month 6, G = {Programmer *B*1}, *A*1 resigned.
- Month 7, G = {Programmer *A*2, Programmer *B*1}, *A*2 replaces *A1*.

Analysis

- At Month 7, *A*2 gets the needs on code P via $A \Rightarrow B \Rightarrow A1 \Rightarrow B1 \Rightarrow A2$
  - To what extent can *A*2 get what to be done, and what options have been tried on P?

Motivation (For Larger Project In Reality) [2/3]

- A2 wishes to know
  - The requirement R that A collected from U (months ago!)
  - The latest changes in R before A2 takes up the position
  - The set of functions $\{f_1, \ldots, f_n\}$ of P that has been identified and designed but not been implemented (in full) with respect to the latest requirements
  - The feedback (from B or U) on the implemented functions $\{g_1,\ldots,g_m\}$ of P with respect to the latest requirements
  - Any known and critical issues in P but not completely fixed
- B2 wishes to minimize his/her workload to pass on the information of P to A2
  - A1 also wishes to minimize the workload to pass info of Q to B2

Motivation (For Larger Project In Reality) [3/3]

- U does not want A, A1, A2 and B, B1 to affect U's daily jobs.
  - Entertaining these A/B is a side business only.
  - Can you expect an estate agent as U to meet with A-A2 and B-B1 instead of meeting with his/her clients (thus, no commissions are received, or a lower salary is the result)?
- Everyone wishes to get quick references and move forwards
- When a project is large, no one knows all the details of the whole project.
  - Waiting for clear instructions and clear task specifications before doing the task is unrealistic.

Needs

- The software project scenario in the previous few slides needs a strategy to control the order of project tasks and the information flow to
  - reduce production overheads,
  - improve information passing efficiency,
  - give tractability from user expectation to delivery,
  - predict and track task/project completion and outstanding issues,
  - improve code quality,
  - track what has been agreed not to design or implement,
  - enable newcomers to be productive as soon as possible, and …
- Otherwise….[next slide]

Bugs and Delays are Ubiquitous:

Tech

China's ride-hailing giant Didi blames 'underlying software failure' for breakdown that plunged service into disarray

- The breakdown at Didi plunged thousands of people into a desperate situation on Monday and Tuesday
- Didi says it will carry out 'in-depth technical risk identification and upgrade work' to fully ensure service stability

CYBER REPORT

The Crowdstrike outage and global software's single-point failure problem

PUBLISHED SAT, JUL 20 2024·11:19 AM EDT | UPDATED MON, JUL 22 2024·11:18 AM EDT

Bugs and Delays are Ubiquitous:

SLIDESHOW

Top software failures in recent history

The biggest software failures in recent history including ransomware attacks, IT outages and data leakages that have affected some of the biggest companies and millions of customers around the world

By Computerworld UK staff, Computerworld | Feb 17, 2020 6:20 am PST

February 2020: Heathrow disruption

More than 100 flights to and from London's Heathrow airport were disrupted on Sunday 16 February, 2020, after it was hit by technical issues affecting departure boards and check-in systems, leaving passengers with little information about their flights and limiting the use of electronic tickets.

August 2019: British Airways (again)

British Airways was struck by yet another IT glitch in August 2019, when system failures caused more than 100 flights to be cancelled and more than 200 others to be delayed.

Facebook, Instagram and WhatsApp

In the first week of July 2019, users across the globe found themselves unable to load photos in the Facebook News Feed, view stories on Instagram, or send messages in WhatsApp. Although Facebook didn't detail the reason for the outage, it did release a

Bitcoin Unlimited

Bitcoin Unlimited suffered a serious memory leak which caused several nodes to fall from 800 to about 300. This is almost 70 percent of the nodes run by Bitcoin Unlimited at the time.

HSBC suffers major outage

Less than a week into 2016, HSBC became the first bank to suffer a major IT outage. Millions of the bank's customers were unable to access online accounts. Services only returned to normal after a two-day outage.

Bugs and Delays are Ubiquitous:
Airport Chaos and System Upgrade

- **Hong Kong Airport**

You are here: silicon.com > Hardware > PDAs

PDAs

Hong Kong airport systems grind to a halt

By Polly Raymond

Published: 9 July 1998 09:37 BST

Computer bugs have brought Hong Kong's airport cargo handling system to its knees.

Show related articles

The $20bn (£12.2bn) airport was only opened on Monday but has quickly stumbled upon system problems, which have caused 24 hour delays for some travellers.

A spokeswoman for the Hong Kong Embassy said: "It's only a minor problem" but was unable to confirm whether systems have returned to normal.

The bugs are in the cargo IT systems and baggage handling operations. Cargo operations have been

Hong Kong's air traffic control system suffers another glitch

Back-up system deployed for first time as operators lose information on positioning and altitude of planes

https://www.scmp.com

Wednesday, July

World: A

Hong Ko

honeym

Hong Kong's new airport cost $19bn to build

Hong Kong's air traffic controllers left flying blind for six minutes as HK$1.56 billion system malfunctions again – but authorities insist safety was not affected

Civil Aviation Department recognises 'importance of incident' as lawmaker Jeremy Tam questions delay in alerting public about the failure

Hong Kong's

built at a cos

two days aft

BBC

Correspondent

Jill McGivering

UK Airport Chaos Due To Software Glitch

By Spyros Georgilidakis | May 27, 2023

Passengers arriving at more than one UK airport faced severe delays because of software prob

control systems.

MTR

Hong Kong / Hong Kong economy

Software error to blame for two-hour Hong Kong MTR stoppage that left tens of thousands stranded

Transport operator says problematic code has been fixed, and improved recovery procedures put in place

SIGNALLING BUG DELAYS MTR'S SHA TIN–CENTRAL EXTENSION MILESTONE

BY JAMES OCKENDEN ON SEPTEMBER 12, 2020

HONG KONG

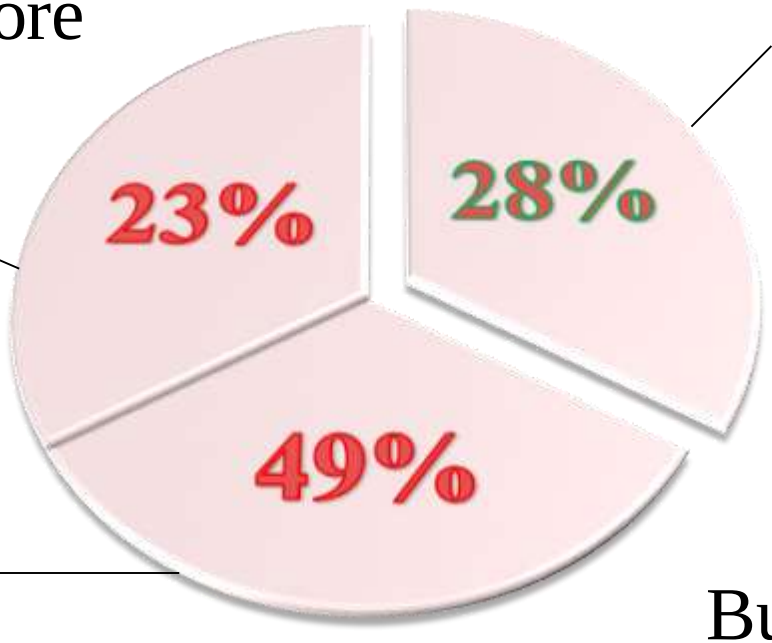
Signal system supplier confirms software issue after Central train crash, MTR Corp says

by KRIS CHENG

Facts (quite some years ago):

Software Development *Project*

cancelled before completion.



completed **on time and within budget**.

cost 89% more than their original budget

But, only **42%** of the original requirements can be implemented.

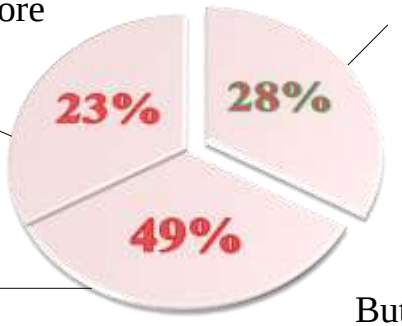
Problems

Nothing can be useful or reused from such cancelled projects?

How to identify a subset of requirements, which are consistent among them, and track their development

Facts:

Software Development *Project*



cancelled before completion.

completed **on time and within budget**.

cost 89% more than their original budget

But, only **42%** of the original requirements can be implemented.

How to streamline the activities better to reduce cost?

Need to prioritize the requirements for guide the development effort

More Recent Findings

- Standish Group's Annual CHAOS 2020 report
  - 66% of projects end in partial/total failure.
    - 31% canceled; 53% with performance degrade
  - Large project: success rate 10%; Small project: success rate 90%
- McKinsey in 2020
  - 17% of large projects go badly
- BCG (2020)
  - 70% of digital transformation efforts fall short of meeting targets.

Recent Situation
Project Failure in 2018


FEATURE

IT project success rates finally improving

After years of stagnating IT project success rates, a new survey from PMI shows that rates are finally on the rise.

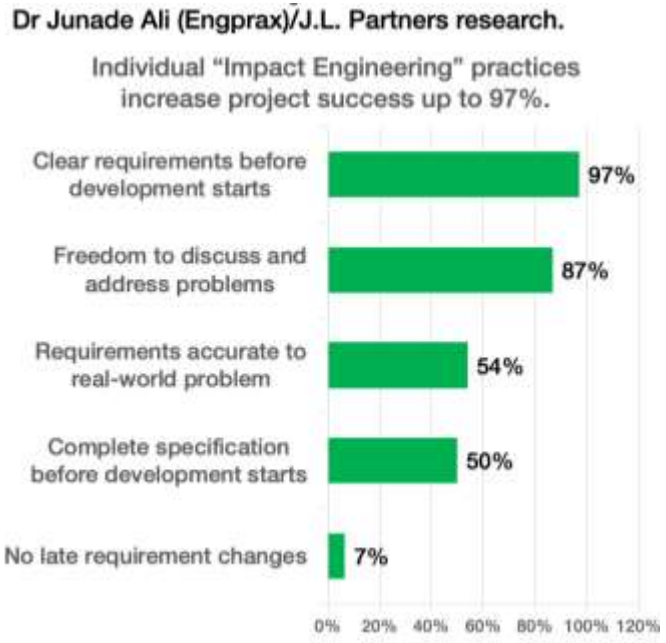
https://www.cio.com

- "The PMI research also identifies a number of factors common to the Champion organizations, says Langley. Besides a focus on **benefits realization**, champion organizations are more likely to focus on **project management talent**, have **at least one PMO** [project management office] within their organization, drive **executive sponsorship** and use **agile approaches to project management**, the research revealed."

https://hbr.org/2023/10/its-time-to-end-the-battle-between-waterfall-and-agile
https://www.engprax.com/post/268-higher-failure-rates-for-agile-software-projects-study-finds

More Recent Findings

- Harvard Business Review (2023)
  - A hybrid of waterfall and agile methods is better
- Impact Engineering (2024)
  - 65% of projects adopting Agile *requirement engineering* practices fail to deliver on time.
  - But if a *more structural and "waterfall"-style* requirement engineering practice is in place, the failure rate for on-time delivery is 10%.



Software Engineering Process

A ***software engineering process model*** is a set of

(1) *activities*,

(2) *techniques*,

(3) *deliverables* and

(4) *tools*

so that software engineers and project managers follow

(5) ***best practices*** to use the items (1)–(4) to manage software systems

SE Process — Activities

- Collecting user requirements
- Designing software
- Coding/Implementation
- Testing
- Deploying the software at user sites
- Maintain software
- Configuration Management
- Project Management

Let’s recall what they are with examples on the next few slides.

SE Process —Activities

Exemplified Software Requirements

User Requirements

Set of Natural Language Requirement

Requirements - Train Station System

Develop a system to model the behavior of a Train-Station. You need to model a train entering the station from the north and then leaving the station to the south. A crossing with boom gates and flashing red lights is located just south of the station. There is a signal to the north of the station that only allows a train to enter when the station is not occupied, that is, when the north signal is green. There is also an exit signal light that ensures the train can only leave the station when the boom gates are down. There is also a north detector that can detect the train approaching the station region from the north. And, there is an exit detector that detects when a train leaves to the south. The following requirements capture the desired behavior.

R1. Initially the station is not occupied. The north signal turns green whenever the station is not occupied. Whenever the north signal is green a train may approach from the north. When approaching from the north a train is detected, by the north detector, which causes the north signal to turn red.

R2. When the north detector detects a train it causes the crossing lights to start flashing red. At the same time, a timer starts timing and when it times out it causes the boom gates to be lowered after which the exit light turns green.

R3. After the train is detected the north detector, it subsequently arrives at the station, the doors open, the people disembark, and then the doors close.

R4. After the doors close the train may leave the station only when and if the exit light is green. When the train leaves the station, heading south, it is detected by the exit detector which means the station is again not occupied. This causes the north signal to turn green and the exit light to turn red. When the exit detector detects the train leaving, it also causes the boom gates to be raised and then the crossing lights to stop flashing red.

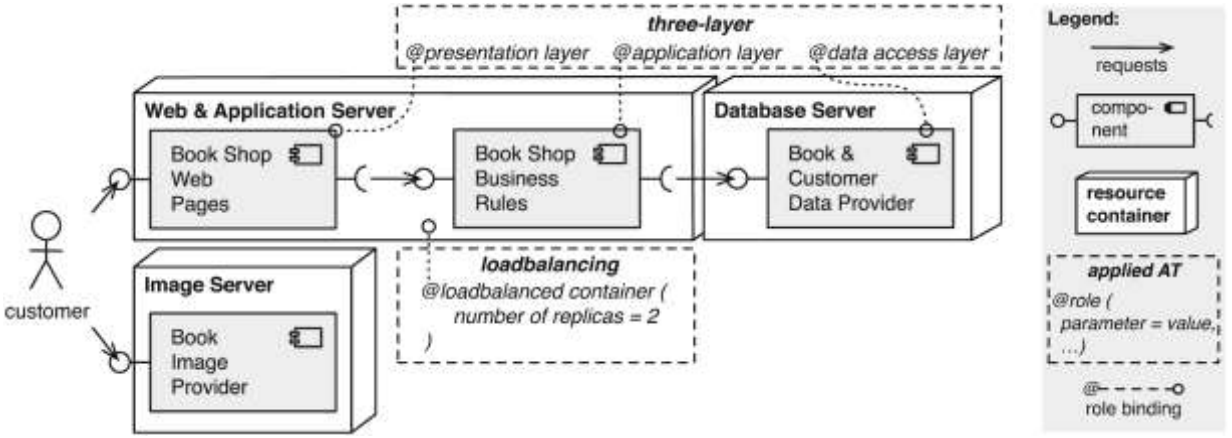
For the purposes of the exercise ignore trains approaching the station from the south. This additional requirement can be integrated later as a separate exercise. Also ignore situations where the train does not stop at the station - this too requires some refinements to the requirements.

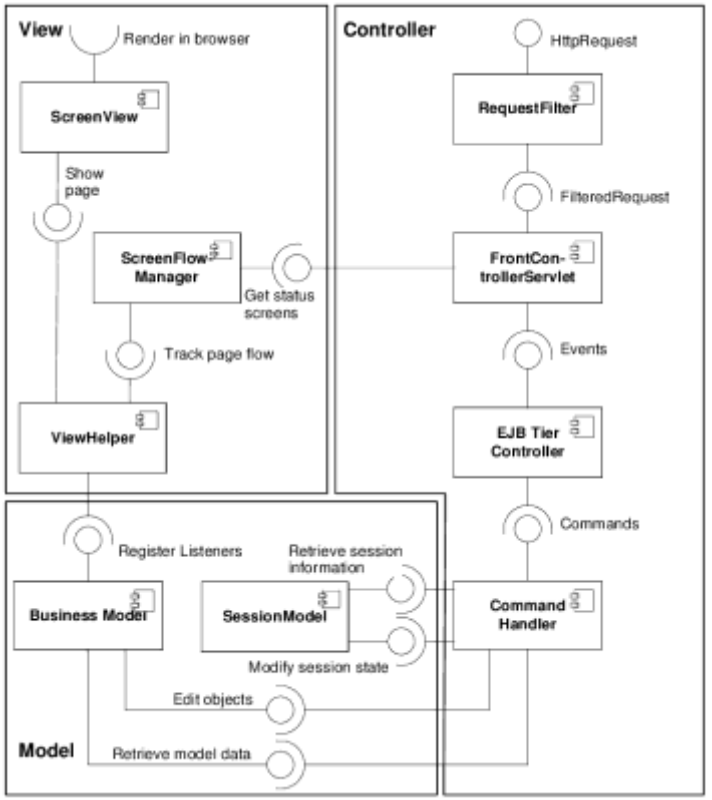
Non-functional Requirements

- Product requirement
  - 4.C.8 It shall be possible for all necessary communication between the APSE and the user to be expressed in the standard Ada character set
- Organisational requirement
  - 9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95
- External requirement
  - 7.6.5 The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system

SE Process —Activities

Exemplified Software Architecture



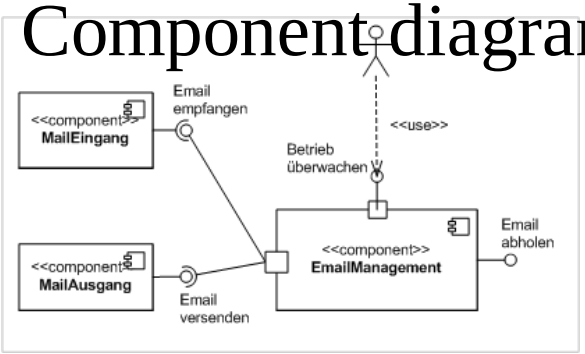


SE Process —Activities

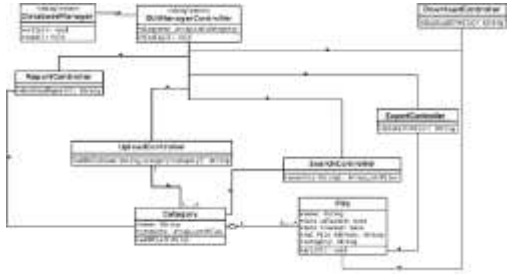
Exemplified Software Designs

Static view:

Component diagram

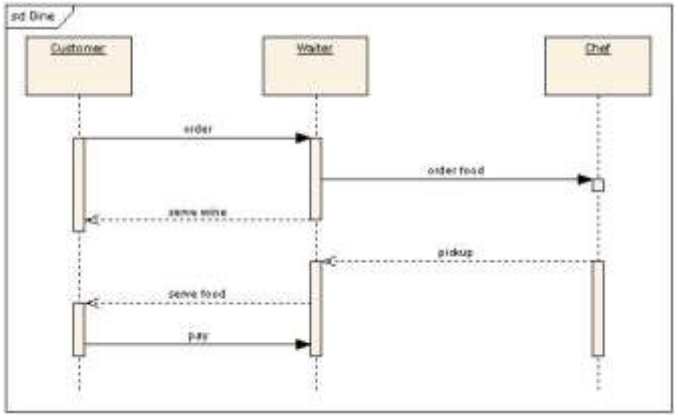


Class diagram

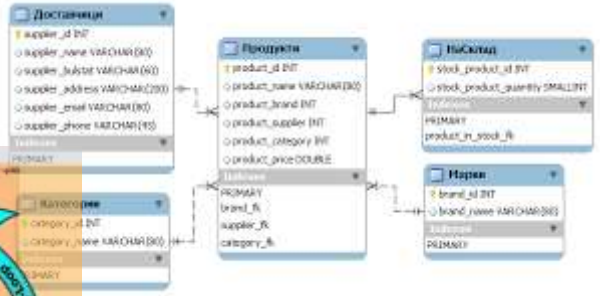


Dynamic view:

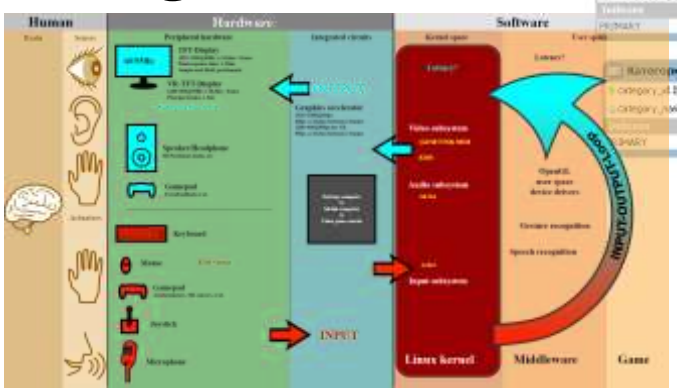
sequence diagram



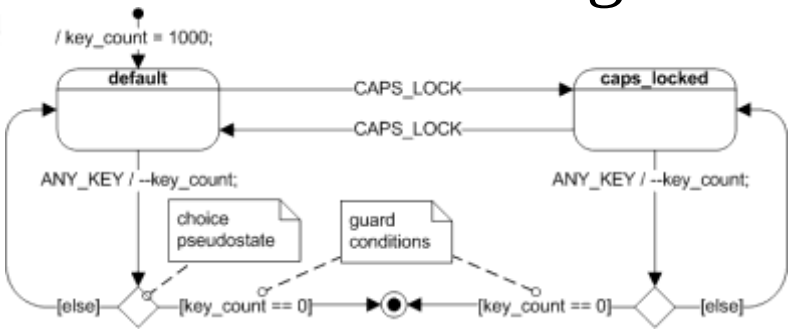
Database scheme



User interface Design

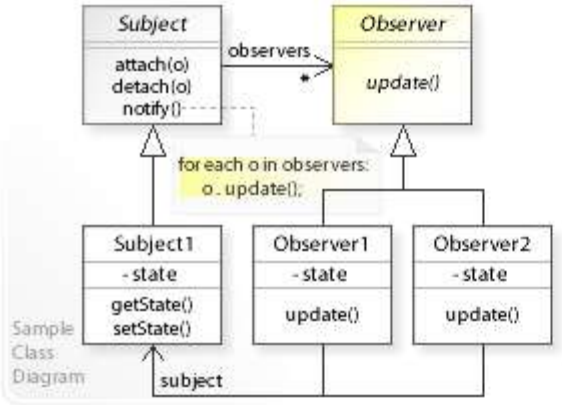


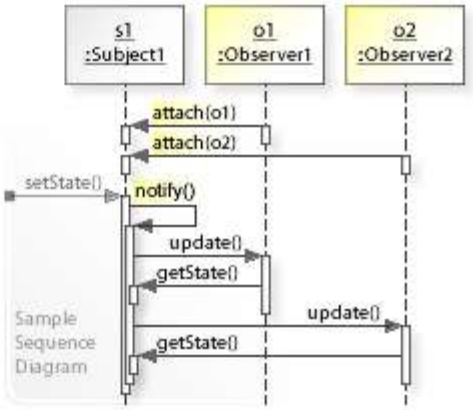
State transition diagram



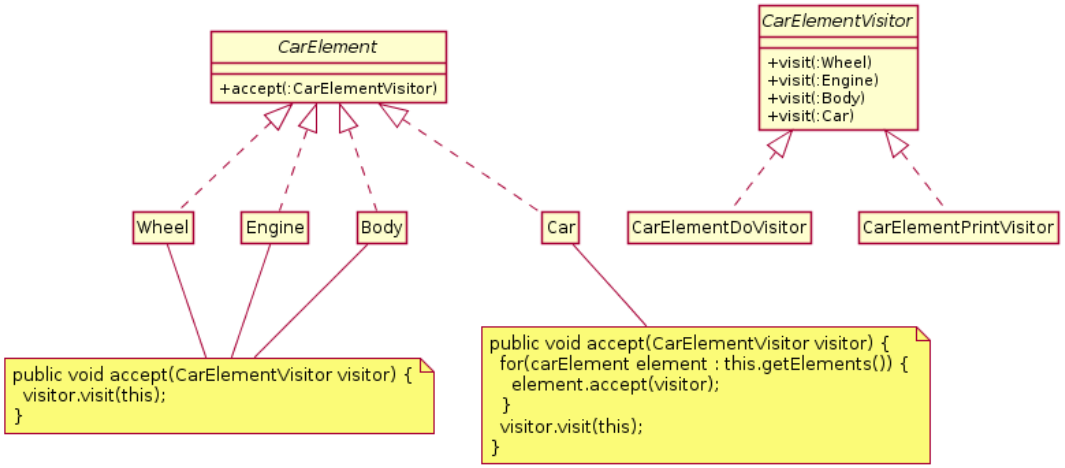
Exemplified Software Design Patterns

observer pattern





visitor pattern



SE Process —Activities

Exemplified Coding with Qulaity

```c
int risky_programming(char *input) {
  char str[1000];

  // ...

  strcpy(str, input);  // Copy input.

  // ...
}
```

better coding

```c
int secure_programming(char *input) {
  char str[1000+1];  // One more for the null character.

  // ...

  // Copy input without exceeding the length of the destination.
  strncpy(str, input, sizeof(str));

  // If strlen(input) >= sizeof(str) then strncpy won't null terminate.
  // We counter this by always setting the last character in the buffer to NUL,
  // effectively cropping the string to the maximum length we can handle.
  // One can also decide to explicitly abort the program if strlen(input) is
  // too long.
  str[sizeof(str) - 1] = '\0';

  // ...
}
```

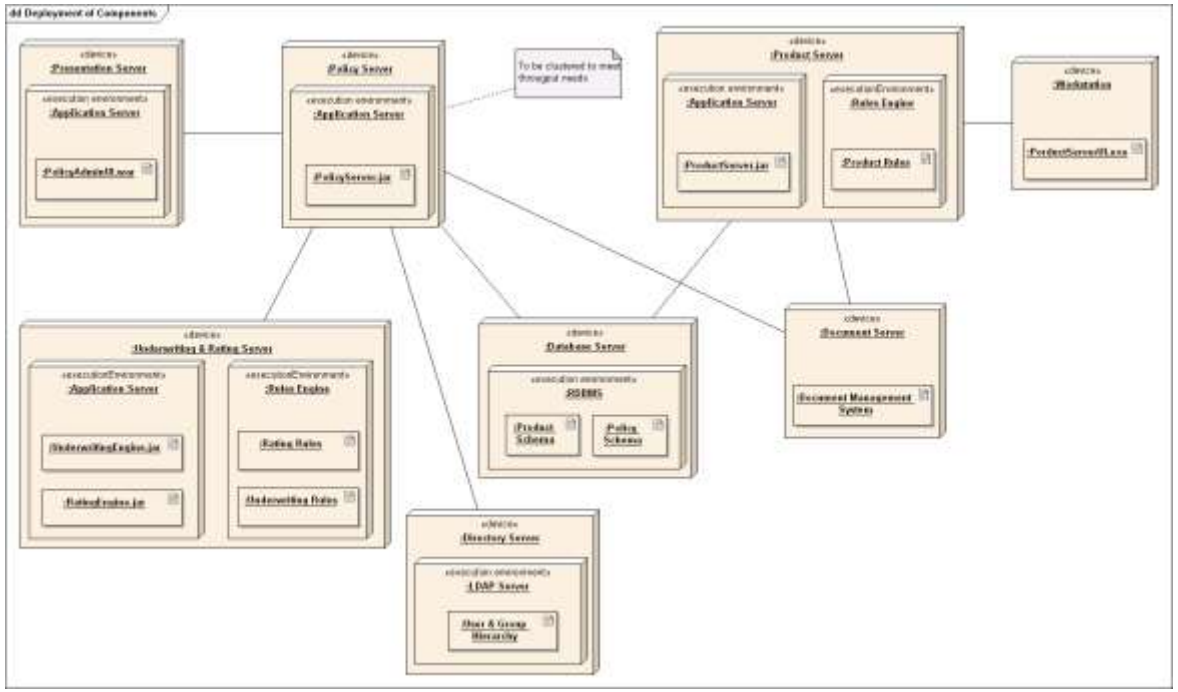
SE Process —Activities
Exemplified Testing

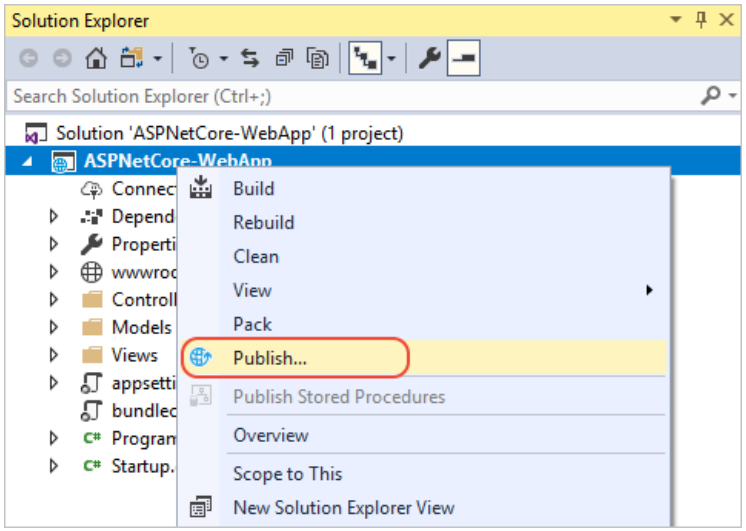
```java
import static org.junit.jupiter.api.Assertions.assertEquals;

import example.util.Calculator;

import org.junit.jupiter.api.Test;

class MyFirstJUnitJupiterTests {

    private final Calculator calculator = new Calculator();

    @Test
    void addition() {
        assertEquals(2, calculator.add(1, 1));
    }

}
```

Unit testing

SE Process —Activities

Exemplified Software Deployment



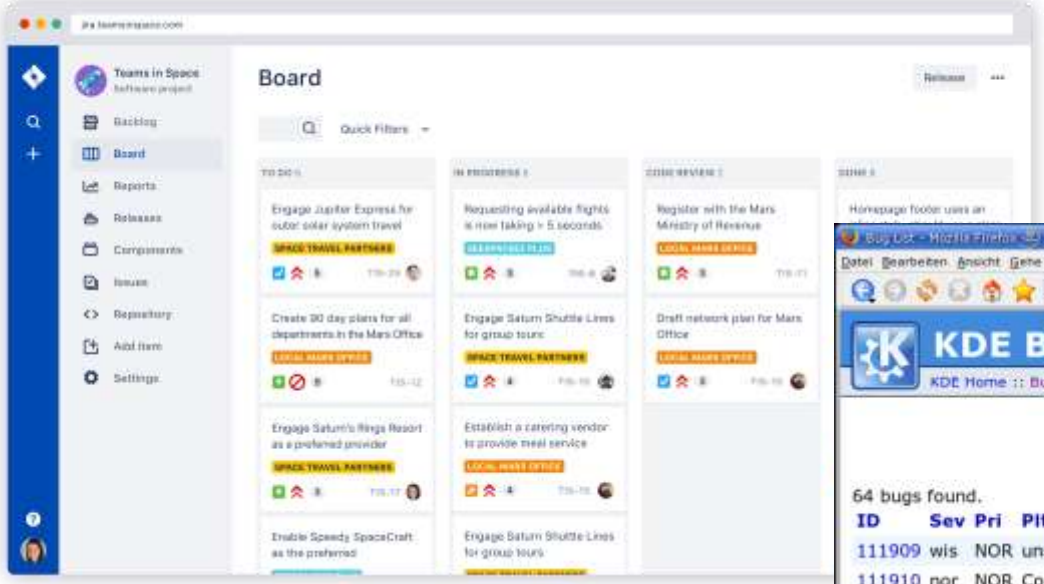


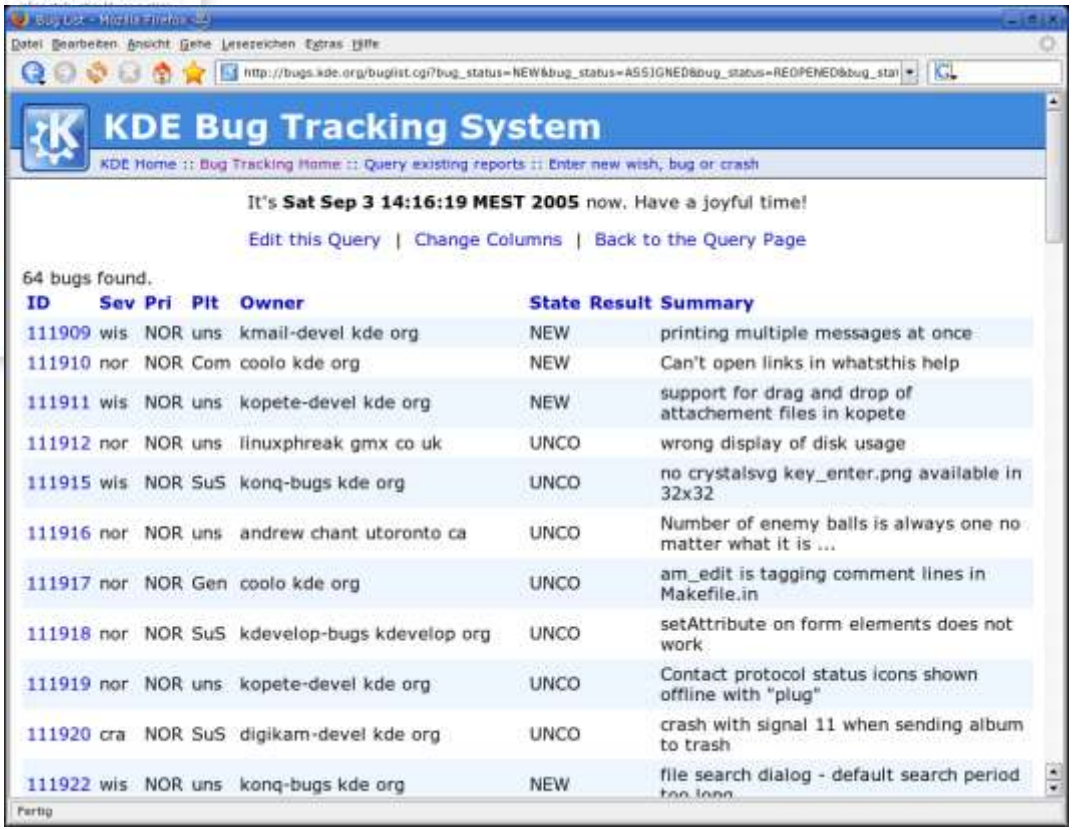
Deployment diagram: Map software component to run on hardware device (or hardware virtualization)

Write scripts or use a tool to publish it

SE Process —Activities

Exemplified Software Maintenance

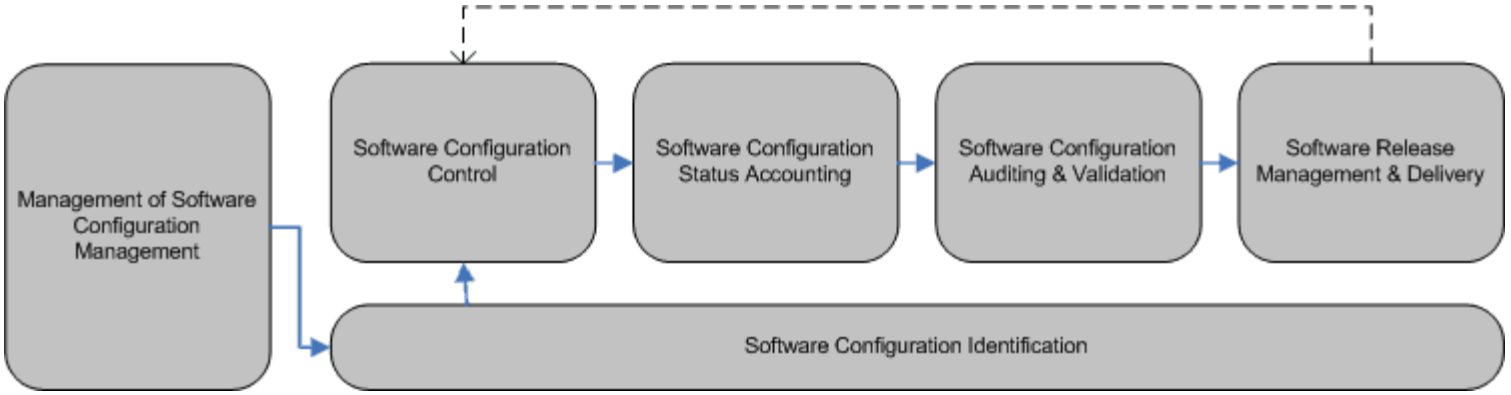




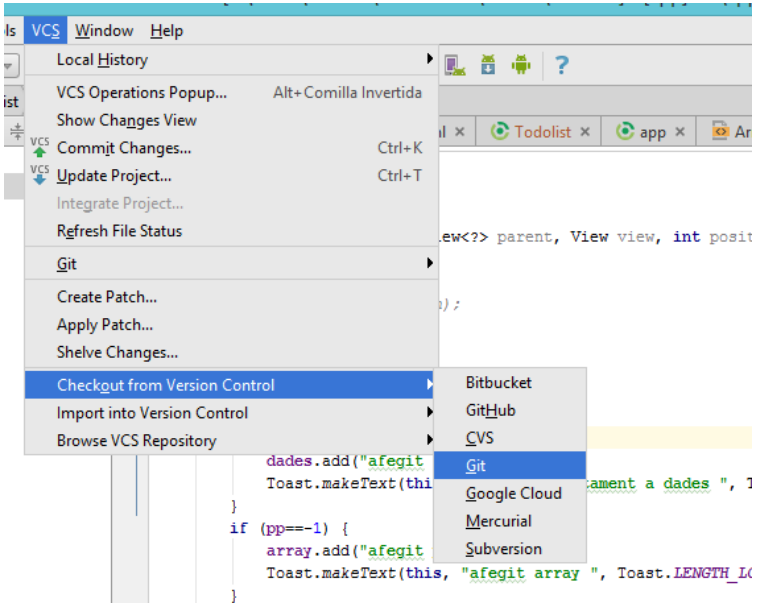
ID	Sev	Pri	Plt	Owner	State	Result	Summary
111909	wis	NOR	uns	kmail-devel kde org	NEW		printing multiple messages at once
111910	nor	NOR	Com	coolo kde org	NEW		Can't open links in whatsthis help
111911	wis	NOR	uns	kopete-devel kde org	NEW		support for drag and drop of attachement files in kopete
111912	nor	NOR	uns	linuxphreak gmx co uk	UNCO		wrong display of disk usage
111915	wis	NOR	SuS	konq-bugs kde org	UNCO		no crystalsvg key_enter.png available in 32x32
111916	nor	NOR	uns	andrew chant utoronto ca	UNCO		Number of enemy balls is always one no matter what it is ...
111917	nor	NOR	Gen	coolo kde org	UNCO		am_edit is tagging comment lines in Makefile.in
111918	nor	NOR	SuS	kdevelop-bugs kdevelop org	UNCO		setAttribute on form elements does not work
111919	nor	NOR	uns	kopete-devel kde org	UNCO		Contact protocol status icons shown offline with "plug"
111920	cra	NOR	SuS	digikam-devel kde org	UNCO		crash with signal 11 when sending album to trash
111922	wis	NOR	uns	konq-bugs kde org	NEW		file search dialog - default search period

SE Process —Activities
Exemplified Configuration Mgt

Process

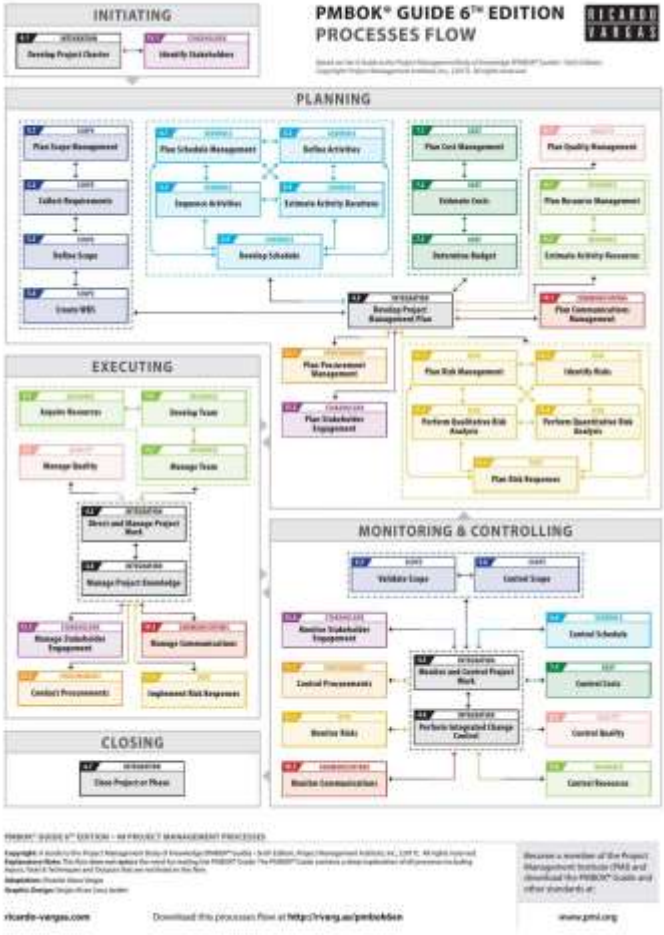


Code version control

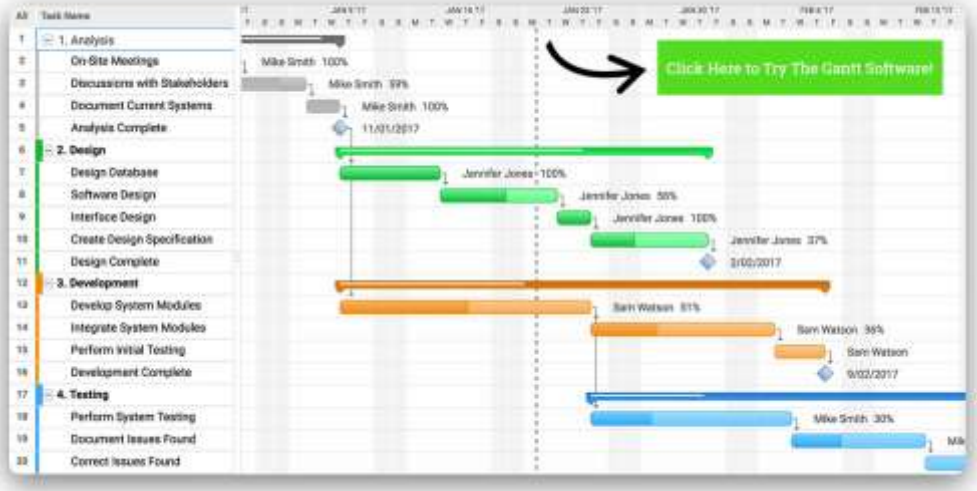


SE Process —Activities
Exemplified Project Management

PMBok



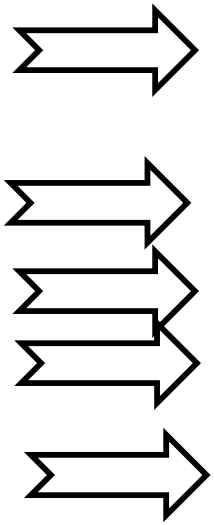
Courtesy https://www.projectmanager.com/gantt-chart



Gantt Chart

Agile-based task board

SE Process — techniques

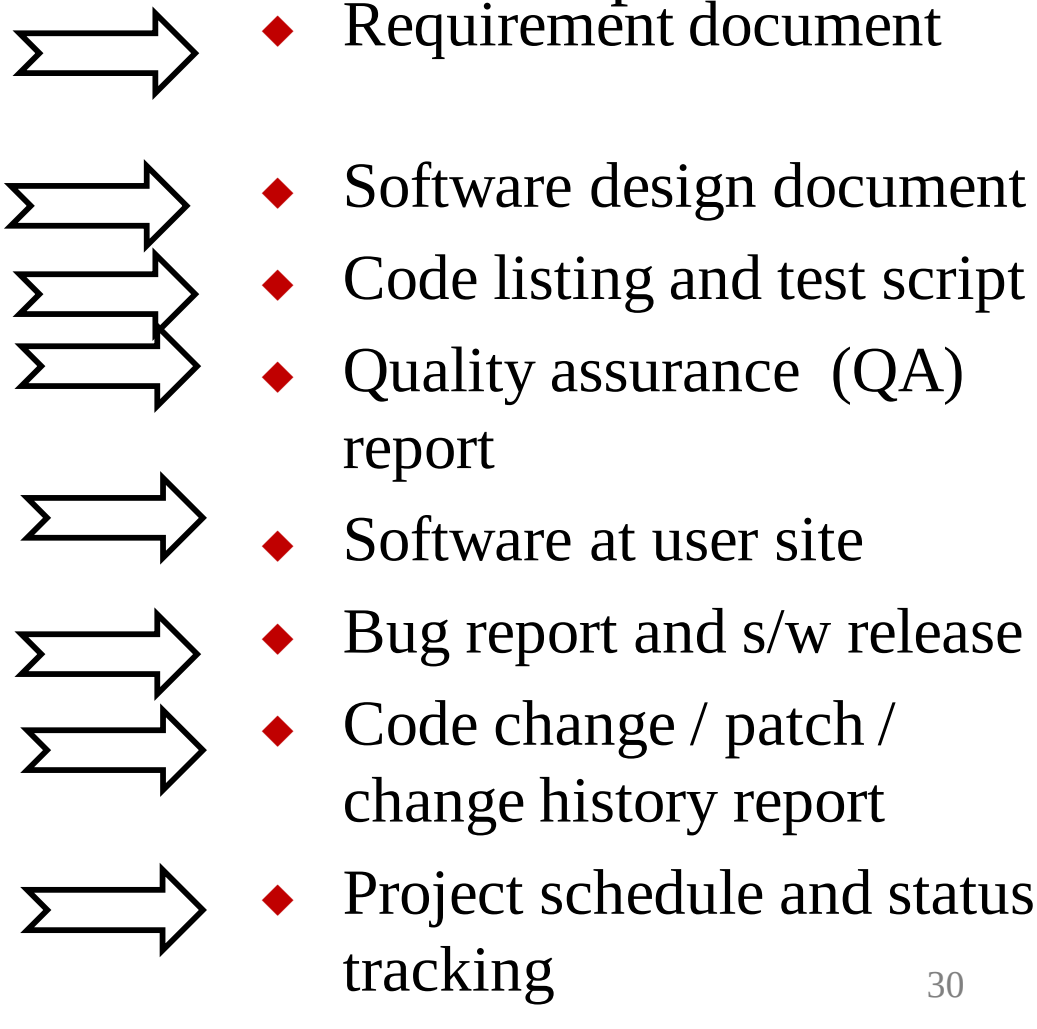


- Collecting user requirements
- Designing software
- Coding/Implementation
- Testing
- Deploying the software at user sites
- Maintain software
- Configuration Management
- Project Management

Example

- Use cases/user stories/use case diagram/meeting
- UML/ patterns/ principles/ tactics
- Java / C++ /framework/platform
- Unit test framework/debugger / …
- Standalone software/Software plug-in/app/web services/…
- Bug reporting/software repository
- Version control/change management software
- Work breakdown structure/ task scheduling algorithm

SE Process — deliverables

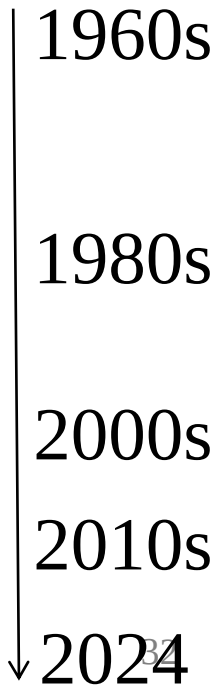


SE Process — tools

	Example
Use cases / user stories / use case diagram	UML tool / Requirement modeling tool / Word
UML/ patterns/ principles/…	UML tool / SonarQube
Java / C++/…	Visual Studio/Eclipse
Unit testing / debugger /…	JUnit/ debugger /fuzzer …
Software plug-in/app/web services	Installer/plug-in framework
Bug reporting/software repository	Bug reporting system
Version control / change management software	Apache subversion / git
Work breakdown structure/ task scheduling algorithm	MS Project/Scrum tool

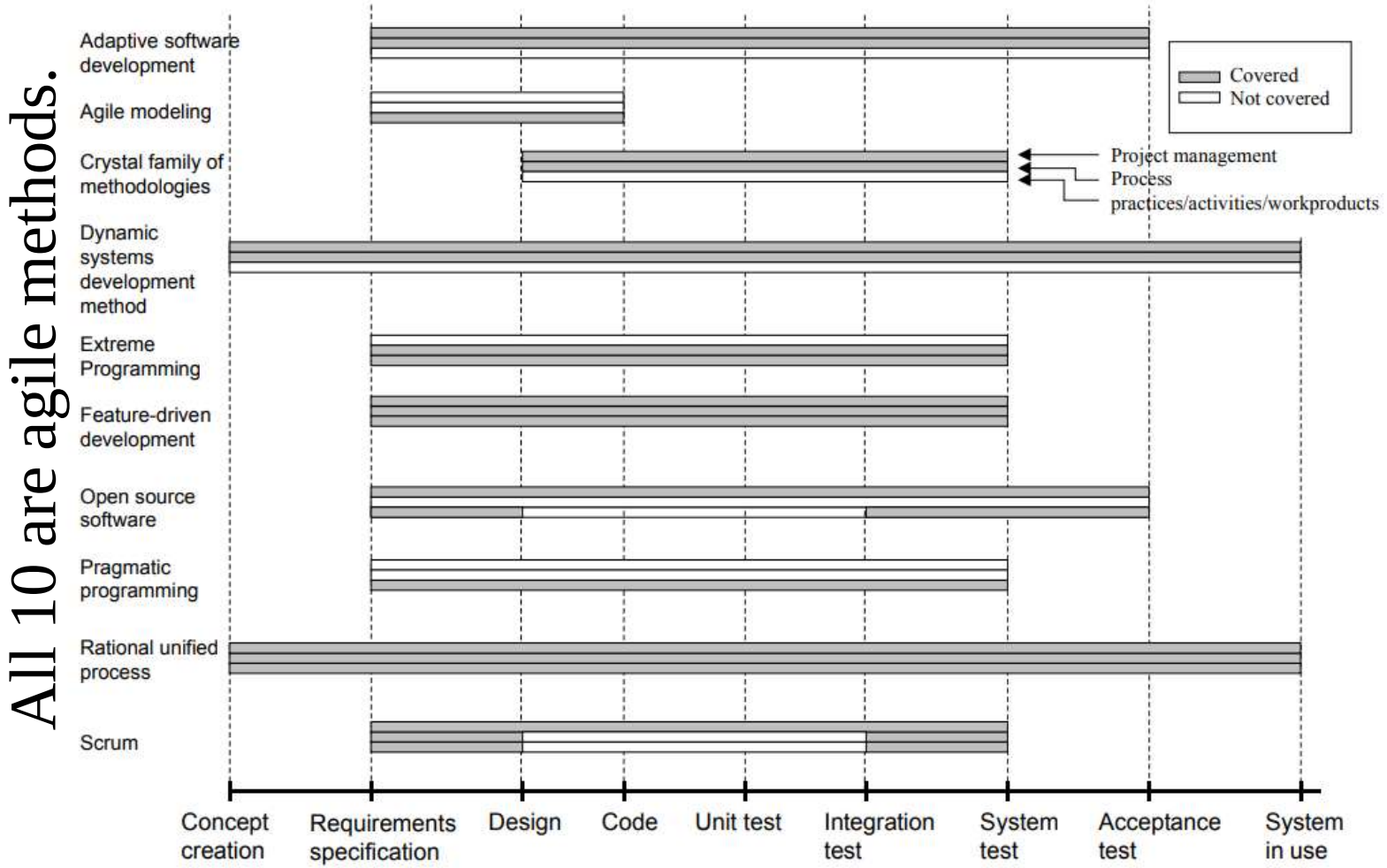
SE Process — Best Practices

- Different ordering of activities defines different types of software processes
- E.g., [see the details in the next few slides]
  - Waterfall model
  - V-shape waterfall model
  - Incremental software development model
  - Prototyping
  - Spiral model
  - Unified Process (UP)
  - Agile Methods
  - Hybrid of Agile Methods with Waterfall



Many Kinds of Agile Methods

All 10 are agile methods.

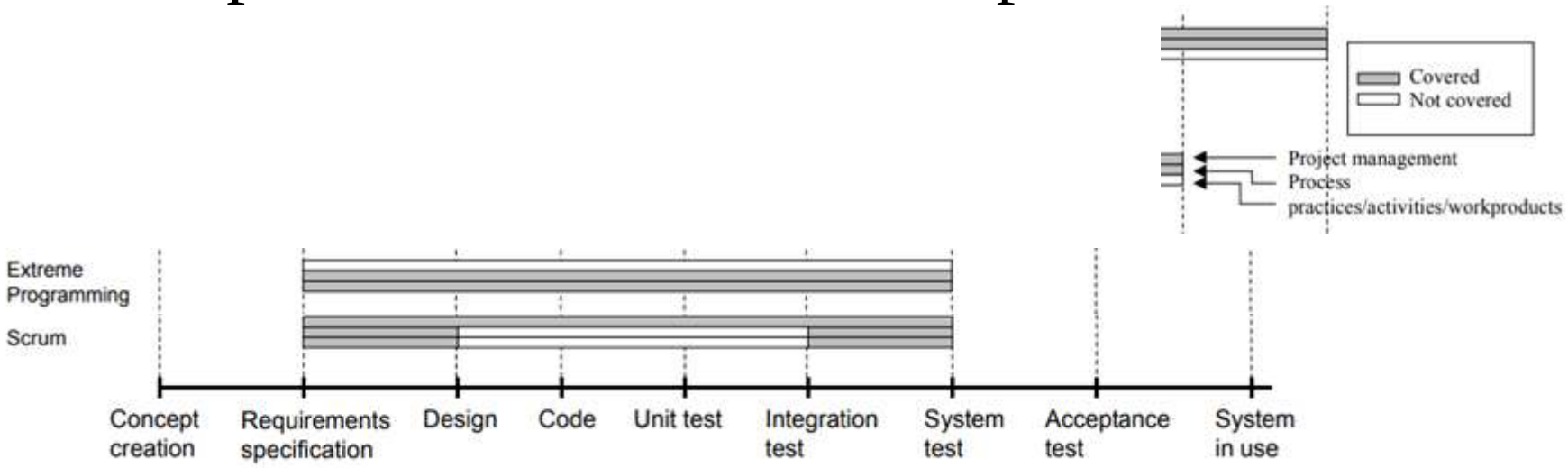


1-Minute Q&A

- **Question 1**: On the last slide, which agile methods include the project management activities?
  - Hint: If an agile method requires fewer specific tools, activities, process steps, and intermediate products, then the agile method is more lightweight.
- **Question 2**: Are there any agile methods, both lightweight and including project management activities? Name them

Extreme Programming vs Scrum

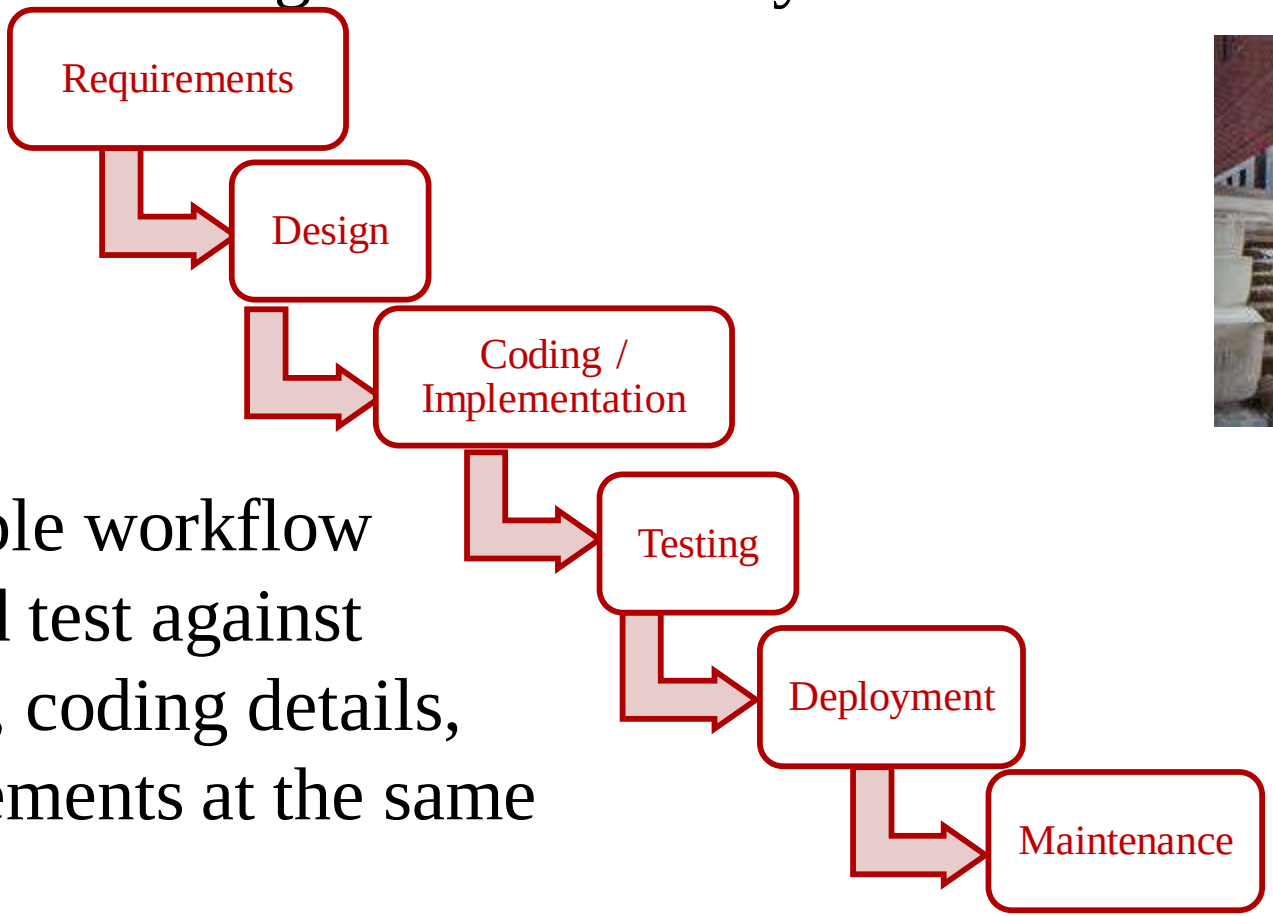
- XP has no PM.
- Scum has PM (but a less tedious process than XP) and other practices/activities/work products



Brief Look on Process Models

Waterfall Model

- *Completely* produce the full set of *deliverables* of each activity before starting the next activity.

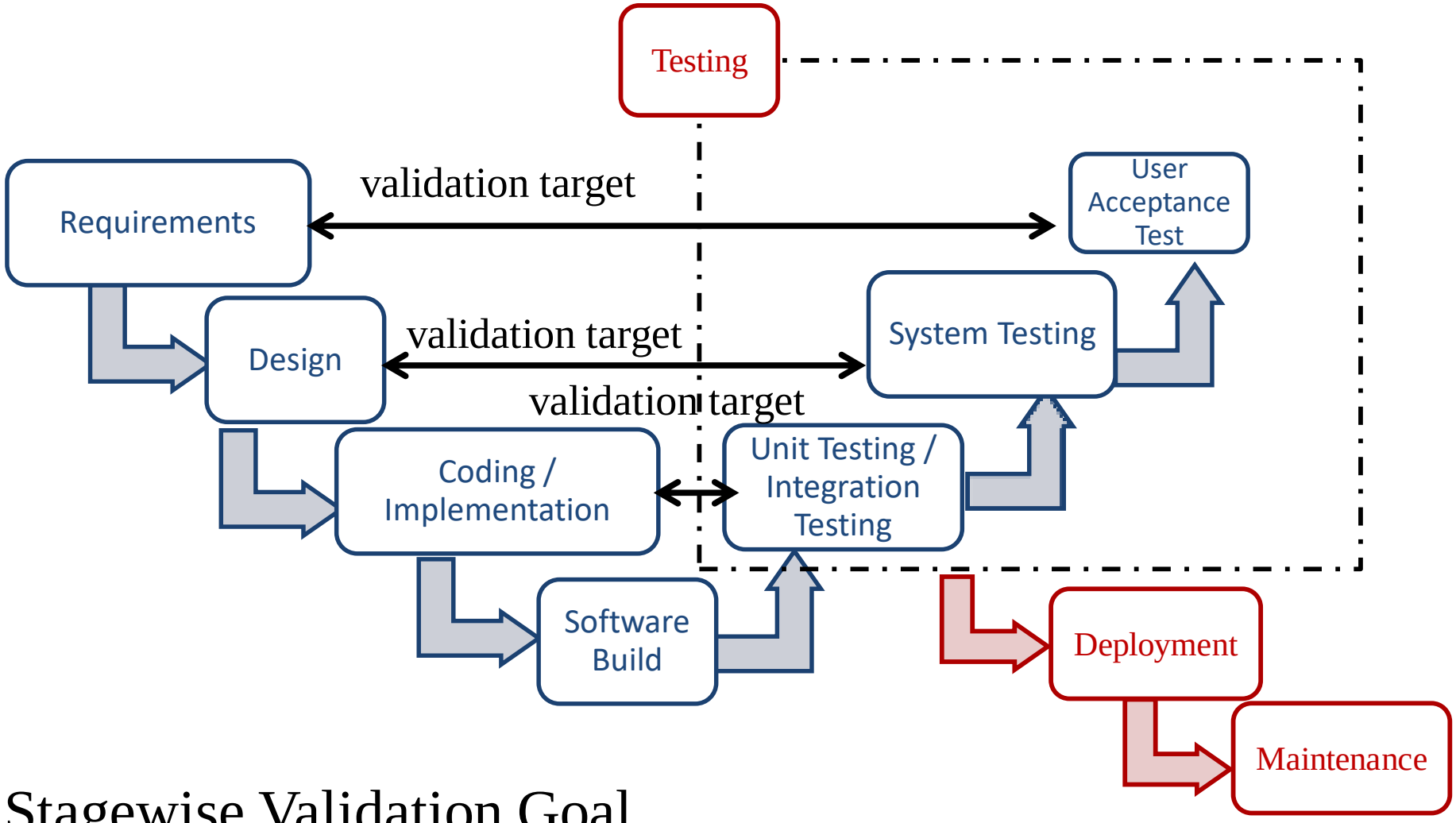


+ Simple workflow
– Need test against design, coding details, requirements at the same time.

Waterfall Method

- Sequential
  - Complete one phase nicely before the start of the next phase.
- Detailed documentation
  - Facilitate everyone comprehensively understanding each phase.
- Predictability in cost and timeline
  - through upfront and high-quality planning
- Rigidity and Incompatibility with change
  - Once confirmed, no change
- Late discovery of issues
- Unsuitability for undefined projects

V-Shape Waterfall Model

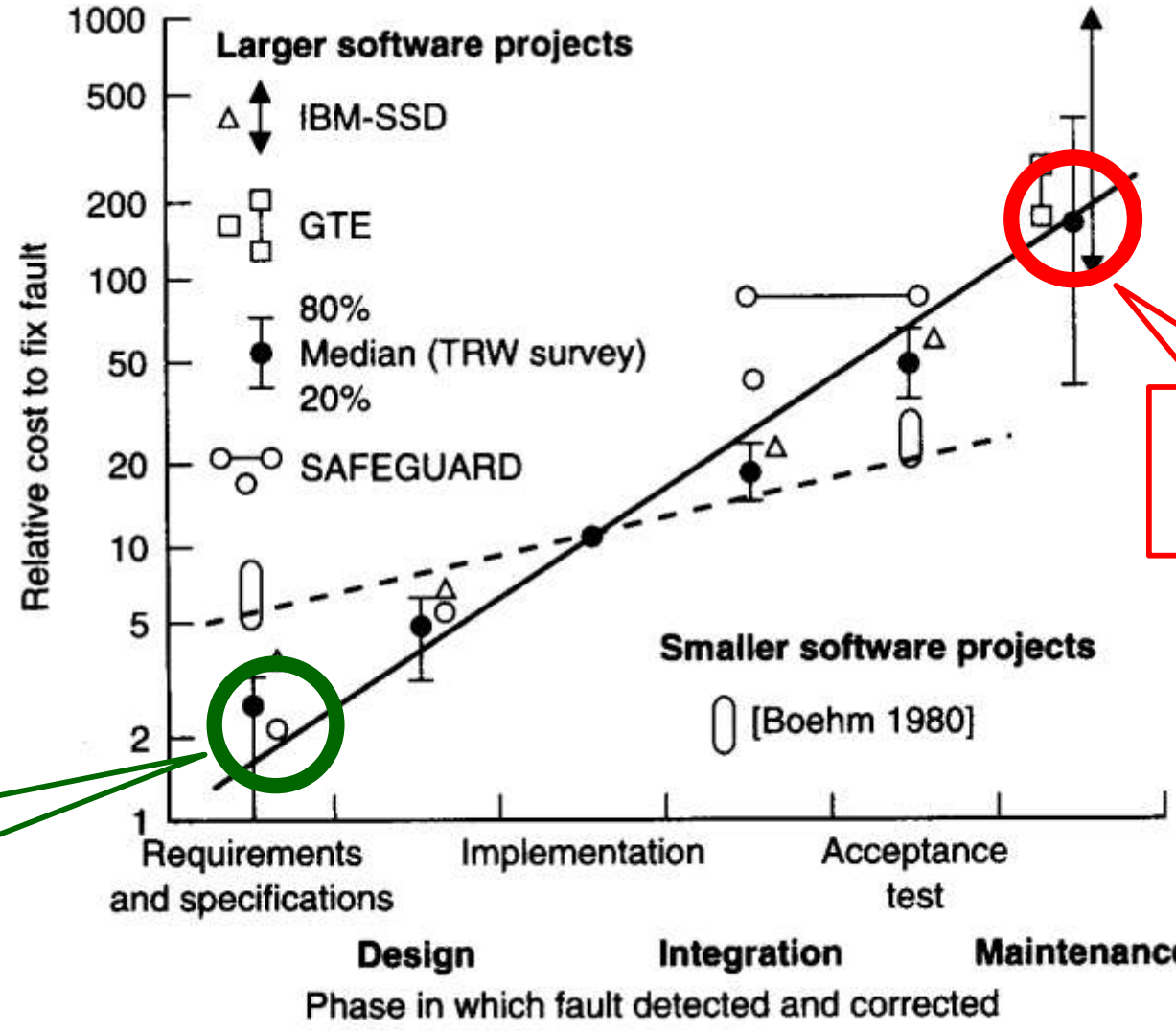


+ Stagewise Validation Goal

Problems of "Waterfall Model" and "V-Shape Waterfall Model"

1. False sense of clear-cut phases of activities.
   - E.g., Can we completely work out the actual requirement before we design our software?
   - It is unrealistic for real-world problems.
   - It is also very costly to correct the problems made in upstream activities. [see the next slide]
2. Nothing is done until they are all done
   - E.g., If we end the project at the design phase, then there is no any code delivered to the project clients, but our clients want to have something executable to help their own business!

Lesson Learnt:
Cost of Fixing Bugs at Earlier Stage is Cheaper



Costly

Cheap

An artifact that can be validated should be validated as early as possible?

Implication on following the Waterfall Model

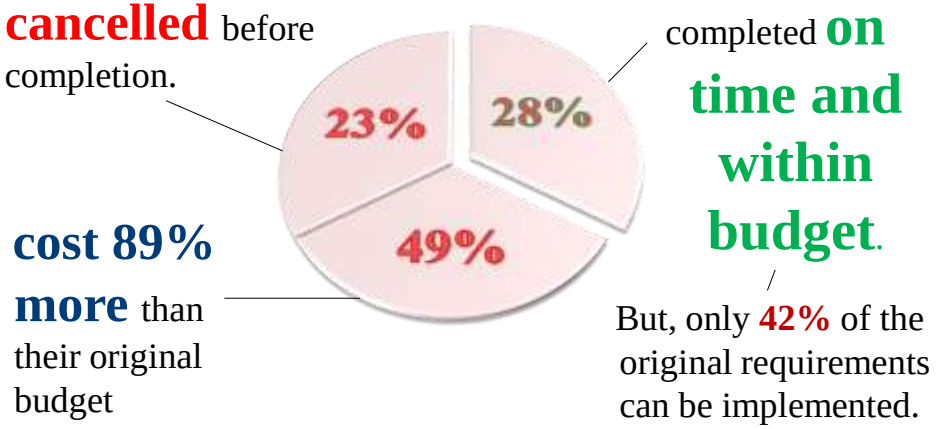
We would rather have a ***gracefully degraded*** **sub**system

- a partial system, a system of lower quality, instead of no system at the end!

An artifact **of an early release subsystem** that can be validated should be validated as early as possible? How?

Facts:

Software Development Project



Software Process Improvement
Current Trends (1)

- Some stages (activities) can be done ***in parallel***
- *Example:* Run technical development and user training in parallel

- Development and QA within each parallel track.
- Each parallel track fits within a self-contained subsystem.
- Quicker for each parallel track to move forward, thereby earlier to rectify bugs ➔ lower cost

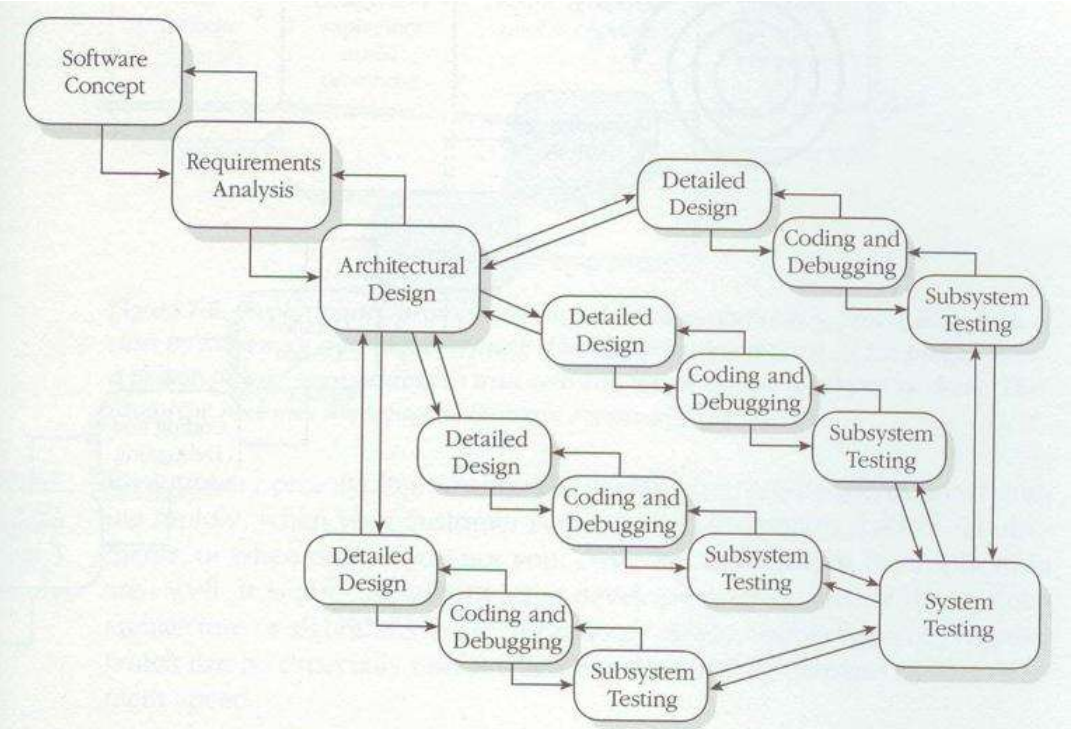
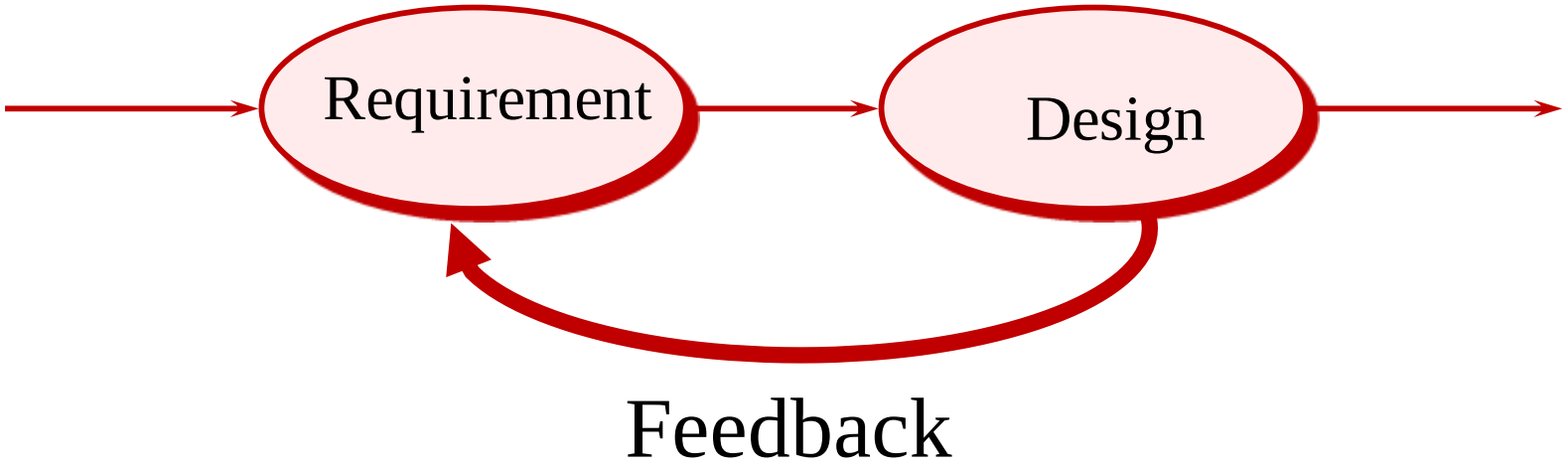


Figure 7-6. *The waterfall model with subprojects. Careful planning can allow you to perform some of the waterfall's tasks in parallel.*

Software Process Improvement
Current Trends (2)

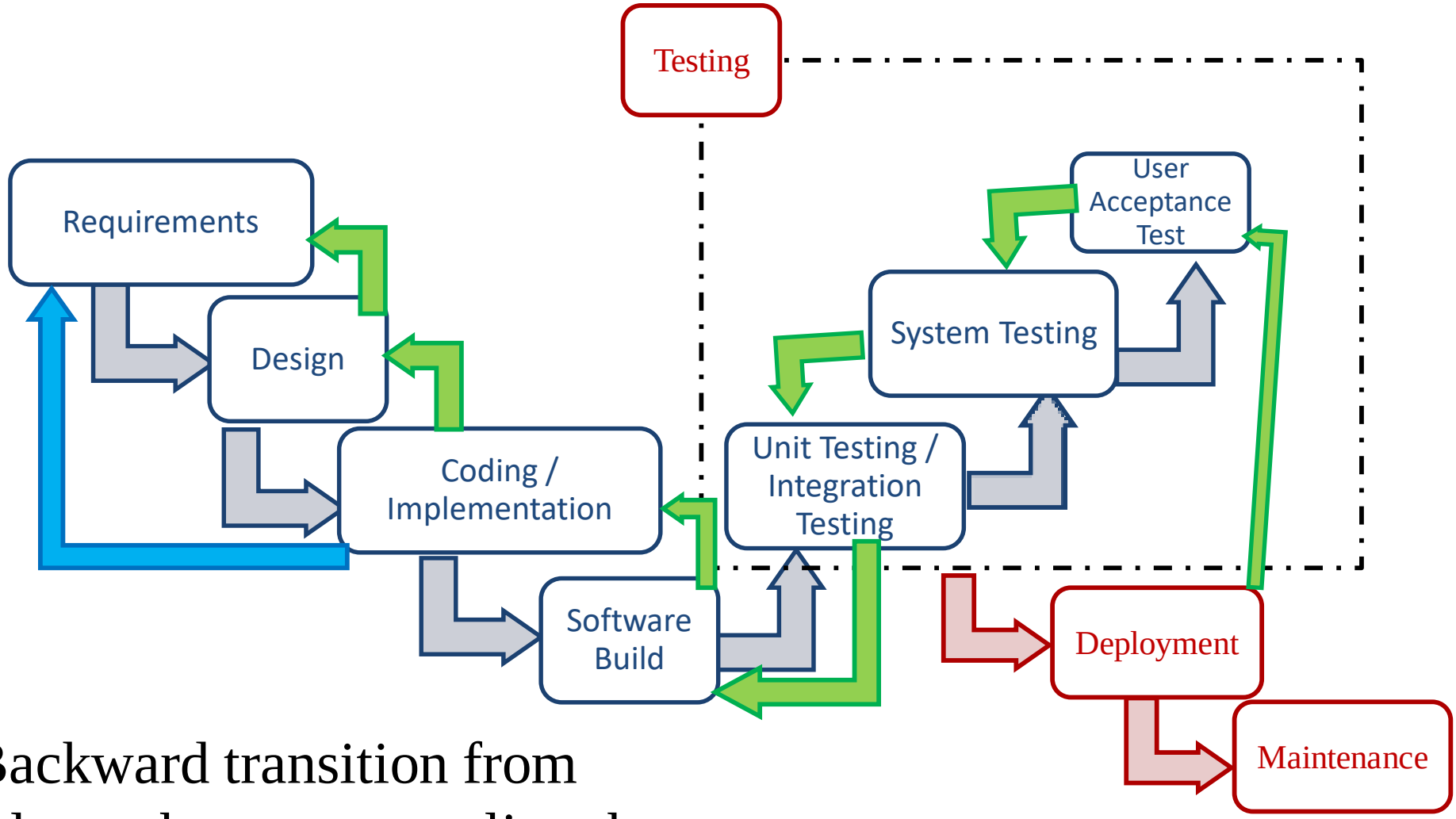
- ***Backward Iterations***

  Example:



- Notice the direction of the arrows.
- Improve earlier stages whenever necessary
- Avoid unnecessary rework or design bugs by working out the more accurate requirements

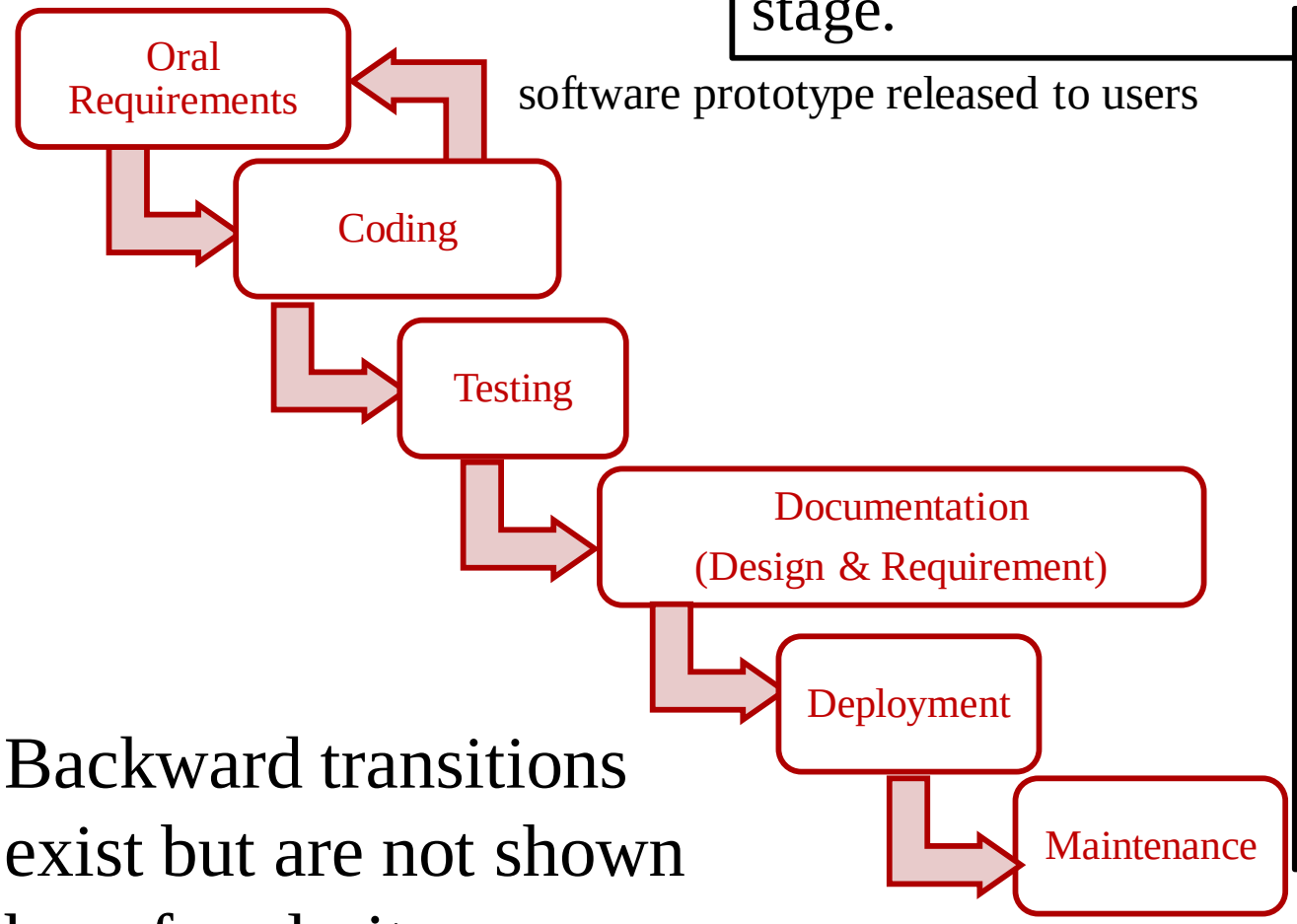
V-Shape Waterfall Model



Backward transition from
a later phase to an earlier phase.

Prototyping

Prototyping = Code before design and requirement engineering. Incrementally obtain and validate the user/system requirements before a proper software design stage.



A prototype is a preliminary version of the final product: buggy with features partially implemented or only with mockup screens so that users can point out their requirements more easily than expressing them more abstractly.

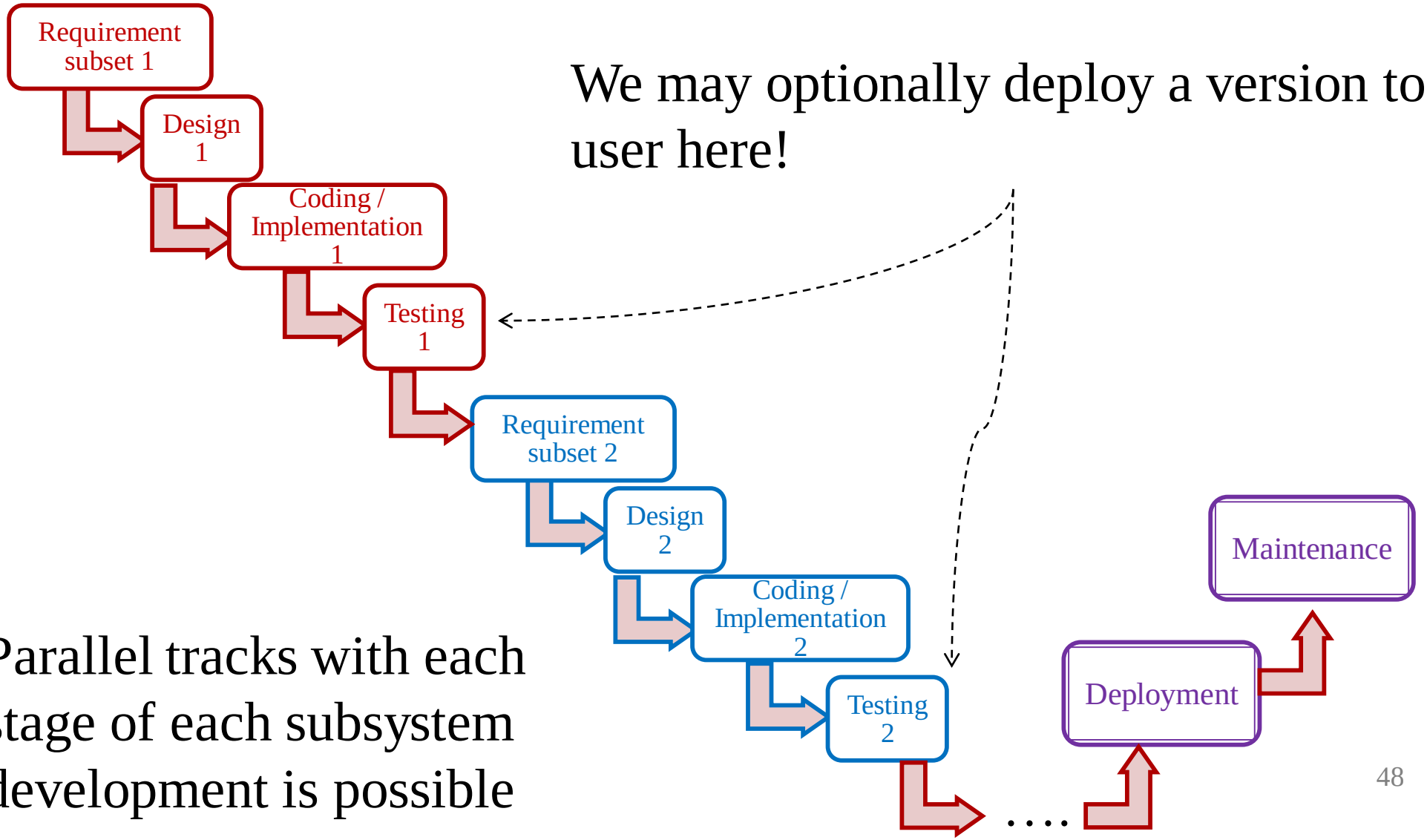
Backward transitions exist but are not shown here for clarity.

Incremental Development

- Rather than using a parallel track to work on a smaller subsystem, another way is to deliver a smaller set of requirements and gradually enlarge the requirement set.
- Divide the set of requirements into subsets.

Implementing one subset to construct a program version ***v*1**
then
Implementing another subset **on top of *v*1** to construct version ***v*2**
then
Implementing yet another subset **on top of *v*2** to construct version ***v*3**
then …
…

Incremental Development

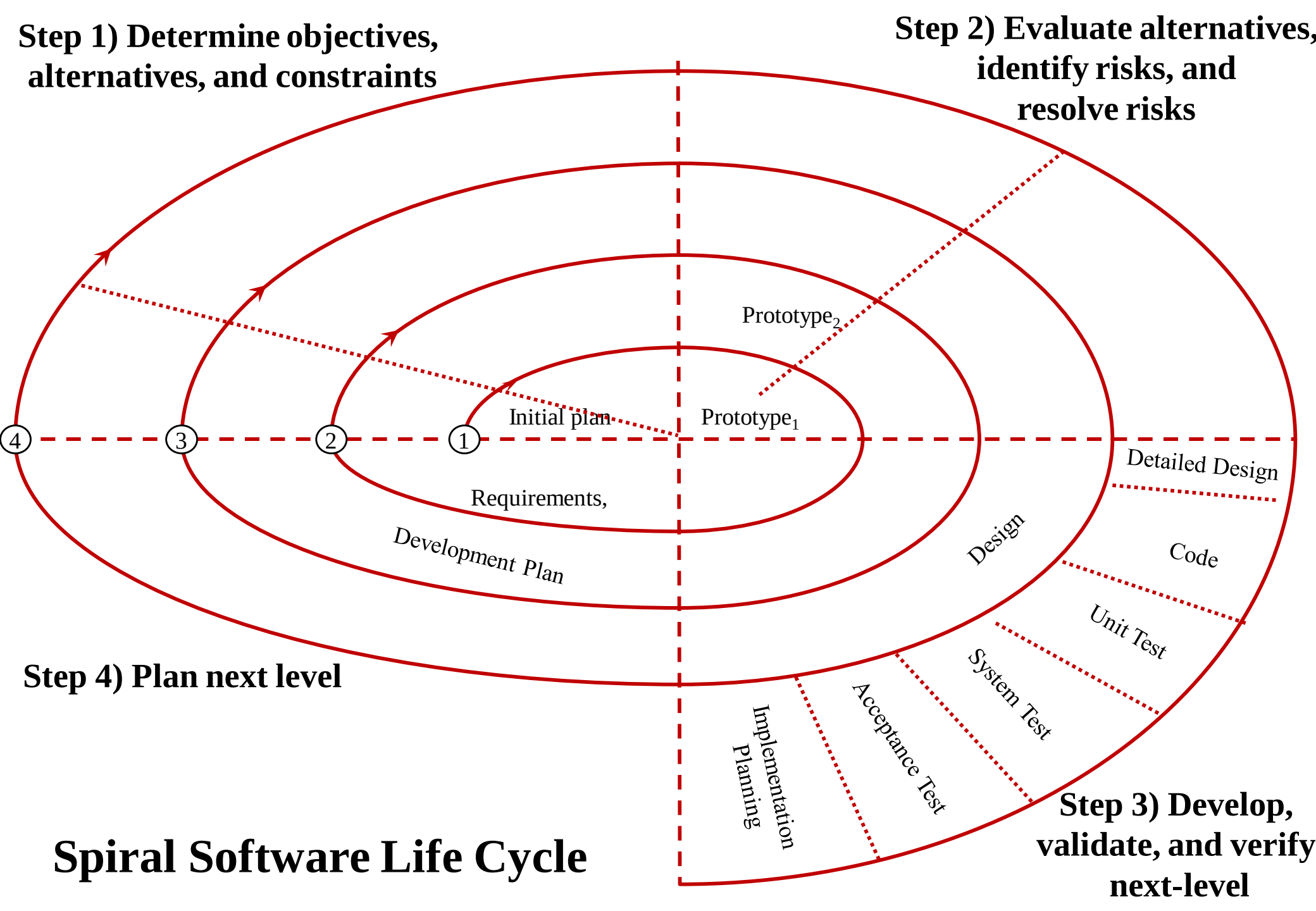


We may optionally deploy a version to user here!

Parallel tracks with each stage of each subsystem development is possible

Spiral Model

- Contains characteristics of the waterfall, prototype, and incremental development, but is more systematic.
- ***Basic idea***: plan, do prototyping, and revise the previous plans until we know the item *X* well through a series of iterations.



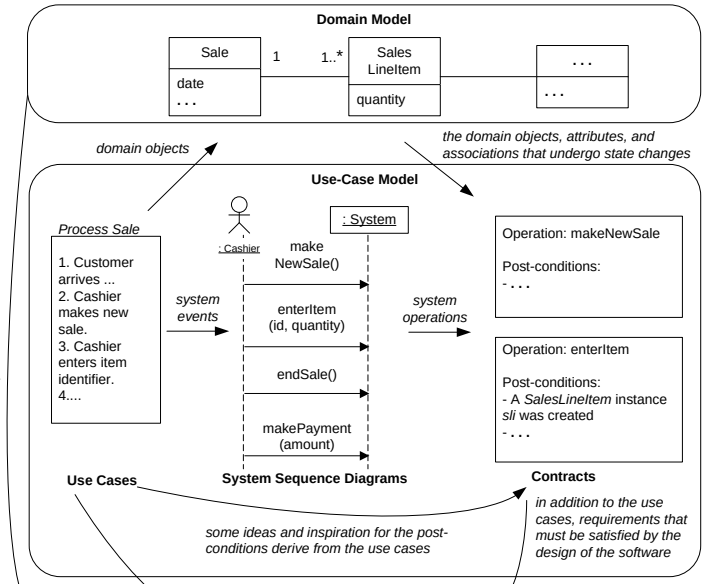
Spiral Software Life Cycle

Software Process Improvement
Current Trends

- ***Model-Based Software Engineering Process*** (various modeling skills as the techniques)

Interact

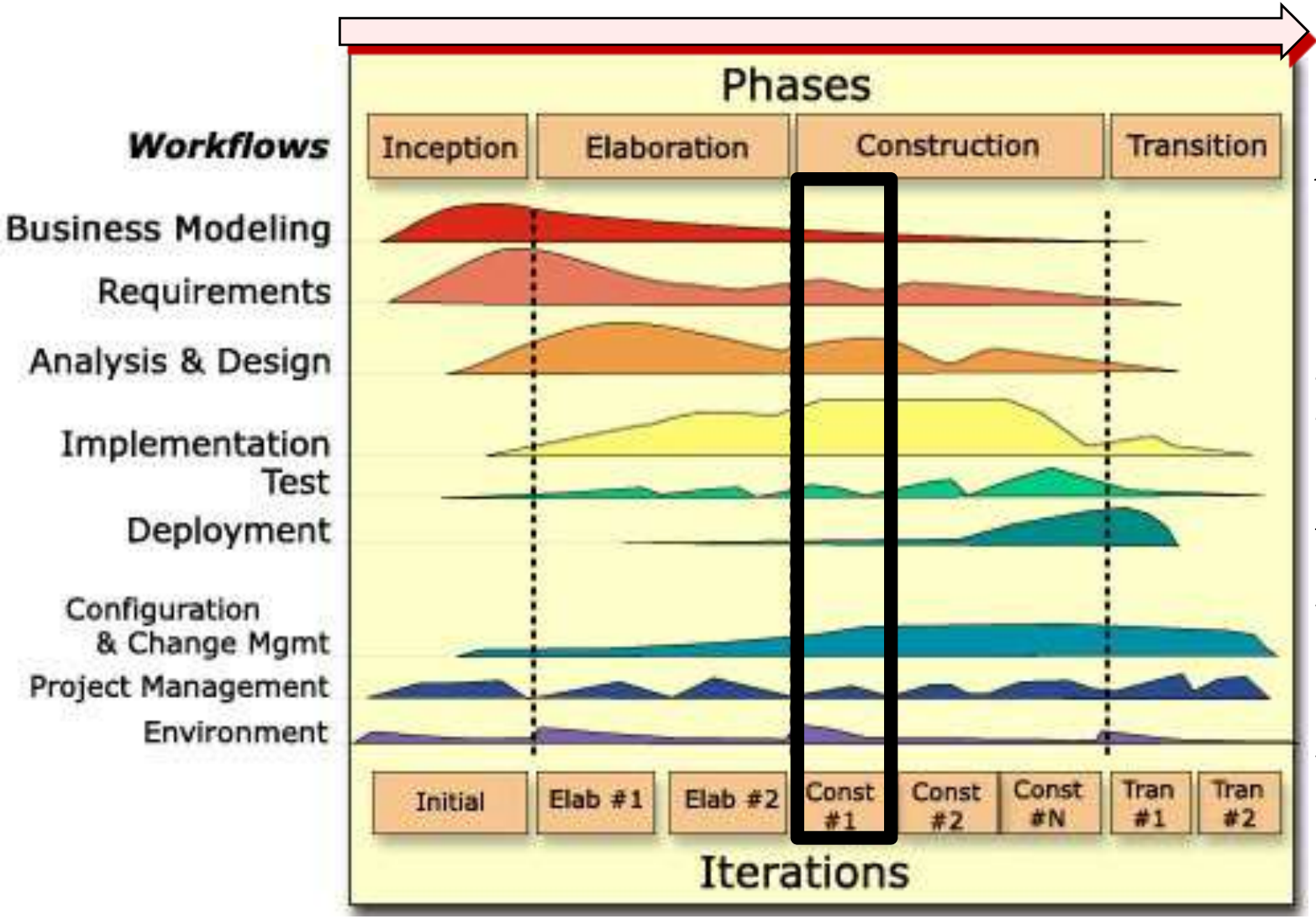
Developer



Construct

Interact

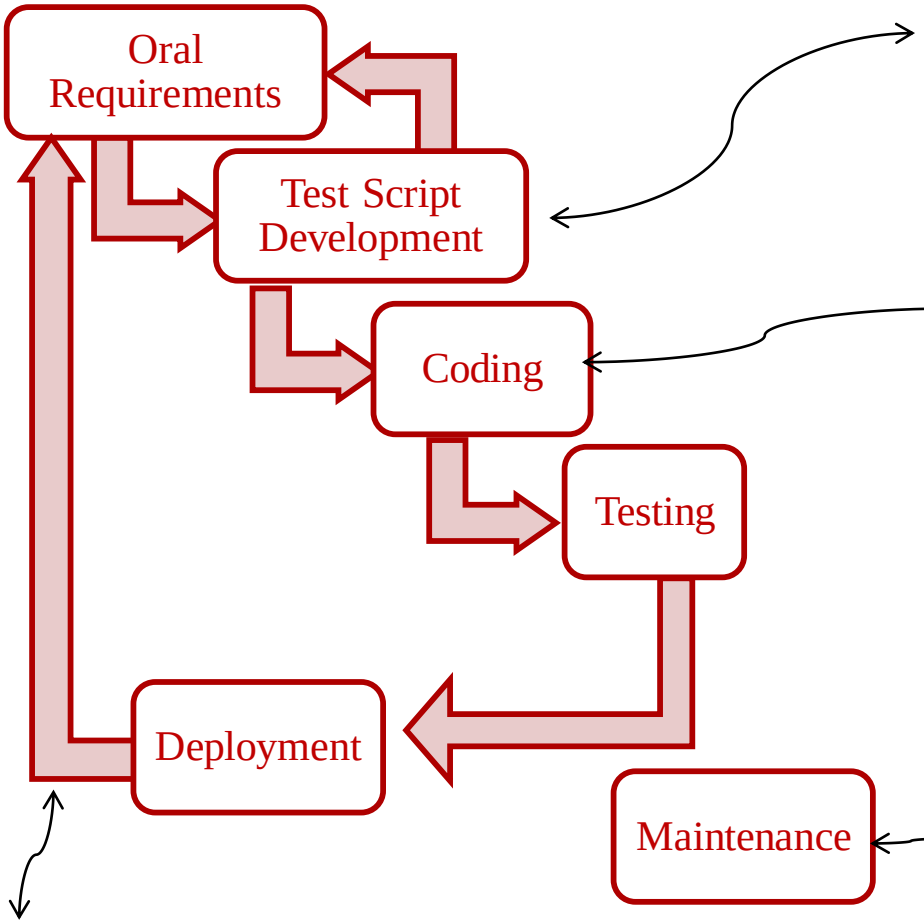
(Rational) Unified Process



Every type of activity **runs in parallel** (vertically) with other types of activities in the same iteration.

All diagrams are rough sketches to ease communications.

Ideas of Test-Driven Development



write automated test case

The minimal functionality of the code that can pass all test cases + applying coding principles (or code refactoring) to keep the codebase better organized.

No obvious maintenance phase.

Each iteration must be short in duration in terms of time.

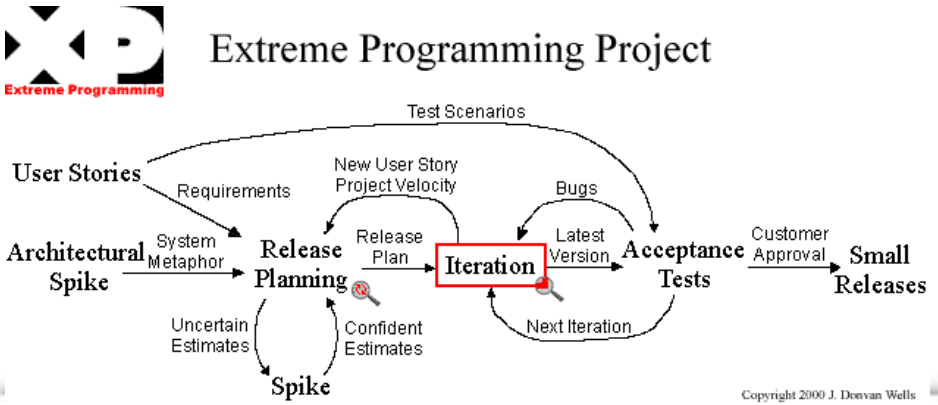
Agile Methods

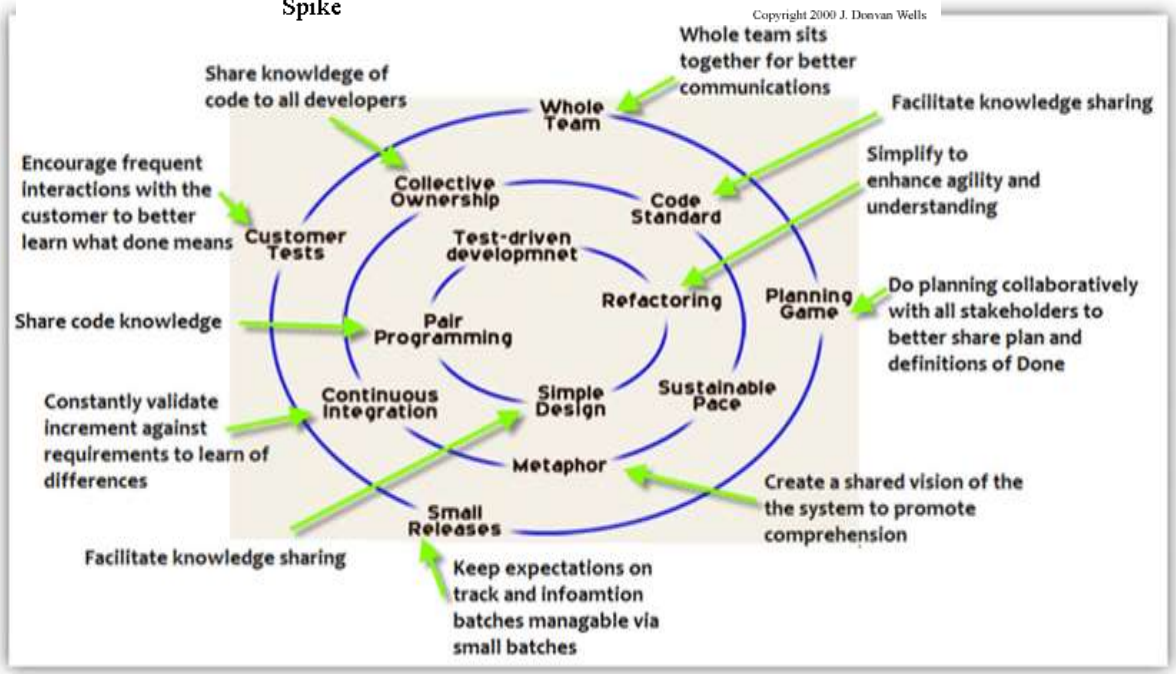
- Iterative development
  - Deliver works in small increments to users
- Customer collaboration
  - evaluate and feedback on the values of requirements/increments
- Response to changes
  - Continuously evaluate the requirements, plan, and deliver increments with user feedback. Make changes rapidly
- Inadequate documentation
- Unpredictable delivery time and costs
- Dependence on client involvement

Example Agile Method: XP

- Extreme programming (XP) has five phases:
  - Planning
    - Meet with users, define user stories, and estimate story points. Plan for release through N iterations (for user stories delivery)
  - Design
    - Use Simple and consistent design sketch
  - Coding
    - Apply XP practice (see the next two slides)
  - Testing
    - Conduct automated unit tests and acceptance tests per user story
  - Feedback
    - PM and users determine the values of the user stories delivered by the implementation

Extreme Programming XP [an Agile Method]





- Emphases:
  - Generate feedbacks
  - Embrace changes
  - keep customers engaged
  - short iteration
  - Fix bugs early

XP practices

The Values of Extreme Programming

Extreme Programming (XP) is based on values. The rules we just examined are the natural extension and consequence of maximizing our values. XP isn't really a set of rules but rather a way to work in harmony with your personal and corporate values. Start with XP's values listed here then add your own by reflecting them in the changes you make to the rules.

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.

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.

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.

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.

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.

What lessons have we learned about implementing XP so far.

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

The Rules of Extreme Programming

Planning

- User stories are written.
- Release planning creates the release schedule.
- Make frequent small releases.
- The project is divided into iterations.
- Iteration planning starts each iteration.

Managing

- Give the team a dedicated open work space.
- Set a sustainable pace.
- A stand up meeting starts each day.
- The Project Velocity is measured.
- Move people around.
- Fix XP when it breaks.

Designing

- Simplicity.
- Choose a system metaphor.
- Use CRC cards for design sessions.
- Create spike solutions to reduce risk.
- No functionality is added early.
- Refactor whenever and wherever possible.

Coding

- The customer is always available.
- Code must be written to agreed standards.
- Code the unit test first.
- All production code is pair programmed.
- Only one pair integrates code at a time.
- Integrate often.
- Set up a dedicated integration computer.
- Use collective ownership.

Testing

- All code must have unit tests.
- All code must pass all unit tests before it can be released.
- When a bug is found tests are created.
- Acceptance tests are run often and the score is published.

XP guidelines

- **Planning**: programmers estimate efforts needed for implementing user stories and customer decides the scope and timing of release
- **Small release**: monthly, or daily for small fixes
- **Metaphor**: A shared story guides all developments by describing how the system works
- **Simple design**: use simplest possible solution
- **Testing**: use tests are implemented before the code. Customer write the functional tests.
- **Refactoring**: do refactoring frequently
- **Pair programming**: two people write code at one computer.
- **Collective ownership**: anyone can change any part of the code at anytime
- **Continuous integration**: integrate the code to the project codebase as soon as it is ready.
- **On-side customer**: customers are available full-time
- **Coding standards**: apply them.
- **Open worksplace:** a large room with small cubicles preferred
- **40-hour week:** No Overtime in two consecutive weeks
- **Just rules:** Team has its own but changeable rules for all to follow

Popular PM elements in Agile Methods

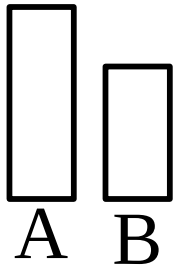
- Estimate the *story point* for each user story
  - Allocate extra time for research on a user story with high uncertainty
- Pick a set of user stories into the next iteration in XP (equivalent to the next sprint in Scrum)
- Within a cycle (an XP iteration or a Scrum sprint), conduct (daily) standing meetings
  - A super short meeting (15 mins in total)
  - [3 mins] Every team member reports
    - (1) what was done yesterday,
    - (2) what will be done today, and
    - (3) what blockers are encountered. For this item, other teammates may help resolve it.
- Measure *project velocity*
- At the end of each cycle, conduct a review (a demo to users about the deliverable achieved in the cycle) and a self-reflection on the cycle

User story

- A user story is in the following format:
  - As a `<role>`, I want an `<action>`, so that I can achieve a `<goal>`.
    - `role` is what we can identify from Requirements Engineering
    - `action` should be tangible (actionable)
    - `goal` should be measurable and quantifiable

    E.g., (for AIMS): As a CityU Student, I want to download a testimonial for my student status by submitting a request in AIMS so that I can apply for my student visa extension through the HKSAR Immigration Department.
- A user story usually has more information such as use cases, diagram sketch, user notes, data, reports from existing systems to clarify the context and scope of the user story.

Relative Story Point Estimation

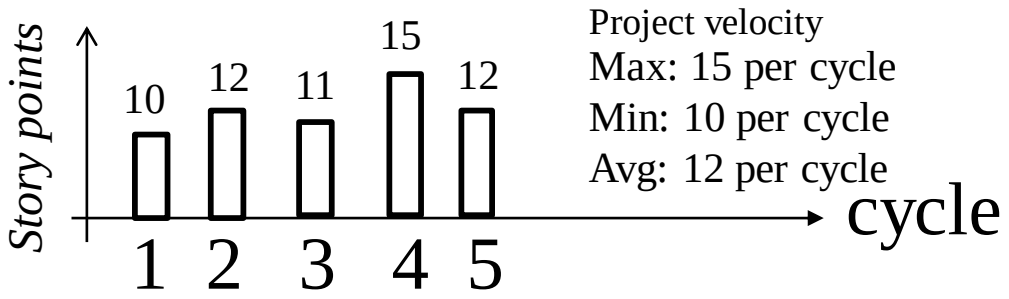


- It is difficult to estimate whether bar A is long and how long bar A is.
- However, it would be easier to say that A is longer than B.

- A story point is a value.
  - E.g., Fibonacci sequence: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …
  - E.g., T-shirt size: XXS, XS, S, M, L, XL, XXL
- Pick one or more user stories as anchors. The whole team agreed on a story point for each such user story to indicate the number of expected efforts to implement the user stories, taking into account uncertainty and implementation complexity.
  - E.g., if B is agreed to as 3 story points, and the team has the impression that A is bigger than B but not very much, A can be assigned 5 story points.

Project Velocity

- A user story is implemented in a cycle if it passes the cycle review (by users and product owner)
- An agile project is organized as a series of cycles (XP iterations or Scrum sprints).
- Each cycle implements a set of user stories with assigned story points.
- If a user story is too large to fit into a cycle, the user story should be broken down into multiple user stories.
- The total story points delivered by the cycle are plotted.

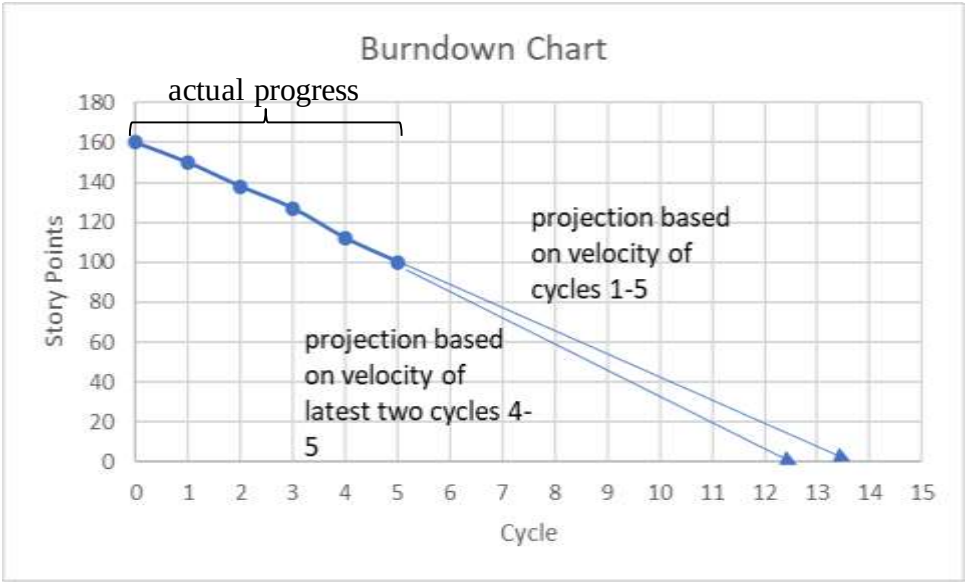


Project velocity
Max: 15 per cycle
Min: 10 per cycle
Avg: 12 per cycle

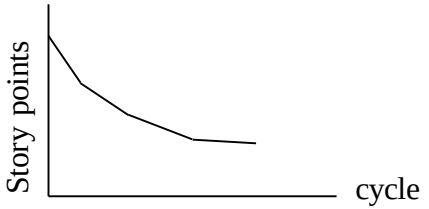
If the outstanding number of story points is 100, then the project needs *ceiling*(100/12) = 9 more cycles to complete

Burndown Chart

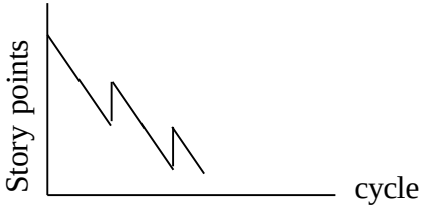
Cycle	Story Points
0	160
1	150
2	138
3	127
4	112
5	100



Problem! The project cannot make progress gradually



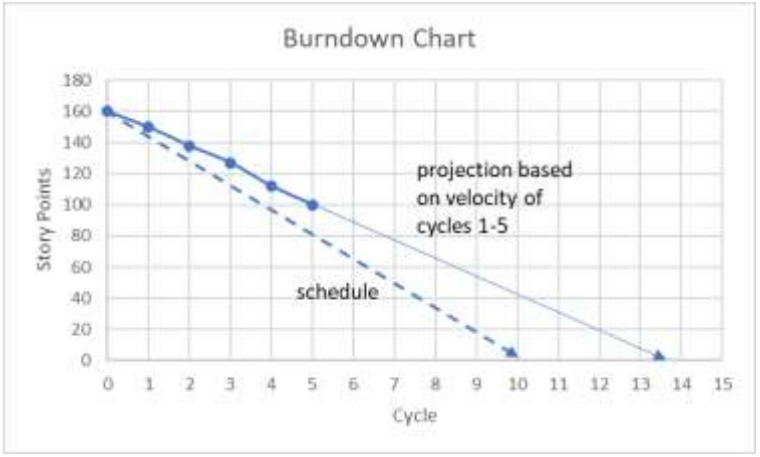
Oops! New user stories added to the project or original estimated story points are underestimated



Project velocity is the slope over a consecutive series of cycles.

Scheduled versus Actual

- When creating a release plan or a plan for a sprint, the project manager needs a schedule.
- Consider our course project (must be delivered by the end of Week 13).
  - Suppose the project starts at Week 2, each cycle being 1 week. The last cycle is to wrap out the project and write a report, prepare a poster or videos, etc.
  - At the end of Week 7 (5 cycles completed), the project is delayed.
    - PM needs to take actions after each cycle: e.g., trim off some user stories and request the team to deliver more story points per cycle



Scrum

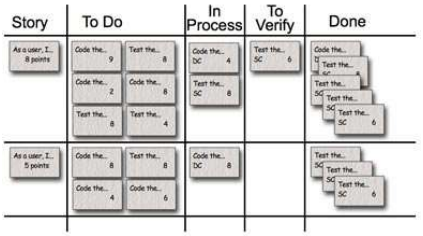
http://en.wikipedia.org/wiki/Scrum_(development)

- Is a kind of agile development process
- Arguably the most widely used development process nowadays
- driven by daily and periodic one-hour meetings
  - Go through the usage scenarios/stories to identify a slice of the highest priority backlog tasks to be completed by the next iteration (Sprint) agreed upon by the customer
  - Together with customers, discuss the goal of the current Sprint, prioritize functions to be completed, and divide into detailed tasks
  - Conduct periodic short planning and review meetings for tasks completed and not completed in time and revise the set of backlog tasks accordingly
- Customers may change their mind at any time
  - Accept that the requirements cannot be fully understood or defined,
  - Focus on maximizing the team's ability to deliver quickly and respond to the selected backlog tasks.

Watch this video: https://tw.voicetube.com/videos/49552 (ignore the ads)

Scrum

Task board



- A product backlog = user stories and development tasks due to the completion of some user stories (e.g., bugs found in a later cycle)
- Priority the product backlog
- Suppose a cycle = two weeks (so, our project has six sprints at most)
- A high-priority subset of the product backlog is treated as the current sprint backlog
  - The size of the sprint backlog is determined by the total number of story points that the team can be delivered within a cycle based on the history
- Through the standing meeting, each team member picks an item from the sprint backlog. The target is to deliver the user story in the cycle. (Visualize as a **task board**)
  - The tasks needed to deliver the story when it is started are all placed into the "to-do" column of the task board.
  - When a task is being worked on, it is moved to the "in-progress" column.
  - When the implementation of the task is completed, it is moved to the "to verify" column (which is usually merged with the-progress column)
- At the end of a sprint, conduct a sprint review
  - Product owners/users verify whether the user story is delivered. If ok, then move the task to the "Done" column.
- Also, conduct a sprint retrospective (reflect on what has been done right to make progress and what to be improved to reduce or avoid blockers to improve productivity, etc)

Scrum Tools and Meeting

https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools

- ***product backlog*** is a list of features desired for a final product, the bugs to be removed, technical work to set up and maintain development environment and user site, and knowledge (e.g., learn to use a new framework) to acquire by the project.
- ***release burndown chart*** tracks progress on a project. The chart itself is updated after each sprint. Teams can measure progress in any unit they choose.
- ***sprint backlog*** is a list of tasks to complete during a sprint. It is updated once a day.
- ***task board*** is a sheet that every member of the team can use and add to over the course of a sprint, and is a visual representation of every task and what phase of completion it's in. Usually, task boards include columns for stories, to-dos, work in process, things needing verification and items finished. Some teams also include burndown charts, notes and tests. ***Hang the board on a wall*** or digitalize it.
- ***Meeting***: The product owner shows up at the sprint planning meeting with the prioritized agile product backlog and describes the top items to the team. The team then determines which items they can complete during the coming sprint. The team then moves items from the product backlog to the sprint backlog. In doing so, they expand each Scrum product backlog item into one or more sprint backlog tasks

Scrum Tools

https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools

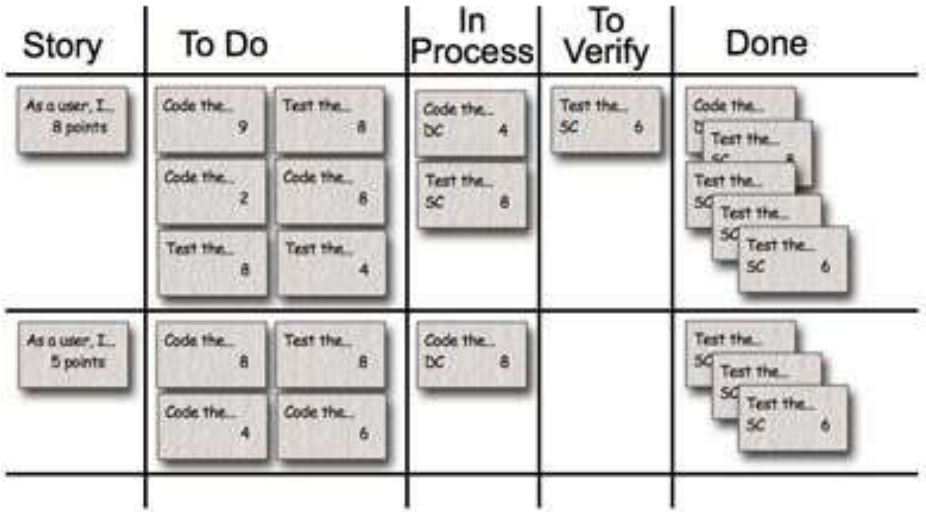
https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools/**product-backlog/example**

Profiles

- As a site member, I want to describe myself on my own page in a semi-structured way so that others can learn about me. That is, I can fill in predefined fields, but also have room for a free-text field or two. (It would be nice to let this free text be HTML or similar.)
- As a site member, I can fill out an application to become a Certified Scrum Practitioner so that I can earn that designation. [Note: Certified Scrum Practitioner was the initial name of what became known as Certified Scrum Professional.]
- As a Practitioner, I want my profile page to include additional details about me (i.e., some of the

release burndown chart

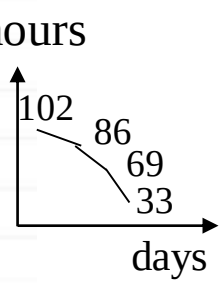
task board



sprint backlog

Hours to go to complete the task

USER STORY	TASKS	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5
As a member, I can read profiles of other members so that I can find someone to date.	Code the...	8	4	8	0	
	Design the...	16	12	10	4	
	Meet with Mary about...	8	16	16	11	
	Design the UI	12	6	0	0	
	Automate test...	4	4	1	0	
	Code the other...	8	8	8	8	
As a member, I can update my billing information	Update security tests	6	6	4	0	
	Design a solution to...	12	6	0	0	
	Write a test plan	8	8	4	0	
	Automate tests...	12	12	10	6	
	Code the...	8	8	8	4	



Sum: 102 86 69 33

Scrum Roles

- ***Scrum Master*** is the team's coach and helps Scrum practitioners achieve their highest level of performance.
  - This role does not provide day-to-day direction to the team and does not assign tasks to individuals.
  - In many projects, this role is assumed by the project manager.
- ***Product owner*** is responsible for prioritizing the backlog during Scrum development.
- ***Scrum development team*** as a whole.

Comparison among Agile Methods

Extreme Programming ⇒

Method name	Key points	Special features	Identified shortcomings
	the method development.	"visionary" and "advisor".	method.
XP	Customer driven development, small teams, daily builds	Refactoring – the ongoing redesign of the system to improve its performance and responsiveness to change.	While individual practices are suitable for many situations, overall view & management practices are given less attention.
FDD	Five-step process, object-oriented component (i.e., feature) based development. Very short iterations: from hours to 2 weeks.	Method simplicity, design and implement the system by features, object modeling.	FDD focuses only on design and implementation. Needs other supporting approaches.
OSS	Volunteer based, distributed development, often the problem domain is more of a challenge than a commercial undertaking.	Licensing practice; source code freely available to all parties.	OSS is not a method itself; ability to transform the OSS community principles to commercial software development.
PP	Emphasis on pragmatism, theory of programming is of less importance, high level of automation in all aspects of	Concrete and empirically validated tips and hints, i.e., a pragmatic approach to software	PP focuses on important individual practices. However, it is not a method through which a system can be

Method name	Key points	Special features	Identified shortcomings
	programming.	development.	developed.
RUP	Complete SW development model including tool support. Activity driven role assignment.	Business modeling, tool family support.	RUP has no limitations in the scope of use. A description how to tailor, in specific, to changing needs is missing.
Scrum	Independent, small, self-organizing development teams, 30-day release cycles.	Enforce a paradigm shift from the "defined and repeatable" to the "new product development view of Scrum."	While Scrum details in specific how to manage the 30-day release cycle, the integration and acceptance tests are not detailed.

⇐ Scrum

Hybrid Method

- E.g., “The video game developer used **Waterfall** to plan and develop assets like characters and initial coding to ensure that the base game would be robust and well-constructed. They **switched to Agile** when it came to gameplay mechanics, debugging, and post-launch updates. By applying a hybrid approach, Ubisoft successfully launched Assassin’s Creed Valhalla to commercial acclaim in November 2020.”

Source: https://hbr.org/2023/10/its-time-to-end-the-battle-between-waterfall-and-agile

Hybrid Method

- Agile PM + Waterfall
- Flexibility and structure
  - Iterative to deliver increments rapidly
  - Adhere to (requirements) documentation
- Phased and iterative
  - Phased approach for well-defined components
  - Iterative approach for uncertain ones
- Customer involvement and predictability
  - getting feedback early and often
- Applicable when (1) Diverse stakeholder needs, (2) Varied project phases, (3) Uncertain requirements, and (4) high-risk projects with complex project structures

Home Reading

- Read the following section of this reference: https://arxiv.org/ftp/arxiv/papers/1709/1709.08439.pdf
  - Section 3.2 Scrum
- Read the differences between Scrum Master and Product Owner in Scrum
  - https://www.yodiz.com/blog/scrum-master-vs-product-owner-differences-in-skills-duties-and-responsibilities-agile-methodology/
- Read the examples mentioned in the Hybrid Method
  - https://hbr.org/2023/10/its-time-to-end-the-battle-between-waterfall-and-agile

References

1. João M. Fernandes and Sónia M. Sousa. 2010. PlayScrum - A Card Game to Learn the Scrum Agile Method. In Proceedings of the 2010 Second International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES '10). IEEE Computer Society, Washington, DC, USA, 52-59. DOI: https://doi.org/10.1109/VS-GAMES.2010.24
   - Include a concise summary of Scrum
2. Videos on YouTube
   - https://www.youtube.com/results?search_query=scrum