

COURSE NAME

SOFTWARE  
ENGINEERING

CSC 3114

(UNDERGRADUATE)

---

## CHAPTER I

# SOFTWARE & SOFTWARE ENGINEERING

---

M. MAHMUDUL HASAN

ASSISTANT PROFESSOR, CS, AIUB

<http://www.dit.hua.gr/~m.hasan>



Google Scholar



LinkedIn

# WHY SYSTEM FAILS?

- ❑ The system fails to meet the **business requirements** for which it was developed. The system is either abandoned or expensive adaptive maintenance is undertaken.
- ❑ There are **performance shortcomings** in the system, which make it inadequate for the users' needs. Again, it is either abandoned or amended incurring extra costs.
- ❑ **Errors** appear in the developed system causing unexpected problems. **Patches** have to be applied at extra cost.
- ❑ **Users reject** the implemented system, lack of involvement in its development or lack of commitment to it.
- ❑ Systems are initially accepted but over time **become un-maintainable** and so pass into disuse.

# SCOPE OF SOFTWARE ENGINEERING

- ❑ The aim of Software Engineering is to solve Software Crisis:
  - Late
  - Over budget
  - Low quality with lots of faults
- ❑ Software crisis is still present over 35 years later!

# SOFTWARE CHARACTERISTICS

- ❑ A **logical** (intangible) rather than a **physical** system element
- ❑ Being **developed or engineered**, but not being **manufactured**
- ❑ Software cost concentrating in **engineering**, not in **materials**
- ❑ Software **does not “wearing out”** but **“deteriorating”** (not destroyed after lifetime like hardware, but backdated by **aging** that needs to update)
- ❑ Software is a **‘differentiator’** (different sub-systems, e.g. **cashier’s workstation in a supermarket**)
- ❑ Without **“spare parts”** in software maintenance (no extra useless features in software)
- ❑ Most software continuing to be custom built (based on the requirements)

# GOAL: COMPUTER SCIENCE VS. SOFTWARE ENGINEERING

- **CS:** to investigate a variety of ways to produce S/W, some good and some bad
- **SE:** to be interested in only those techniques that make sound economic sense

# SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

- ❑ A structured set of activities required to develop a software system
- ❑ The way we produce software, including:

1. Requirements Analysis
2. Designing/Modeling
3. Coding /Development
4. Testing
5. Implementation/Integration phase
6. Operation/Maintenance
7. Documentation

# GOOD & BAD SOFTWARE

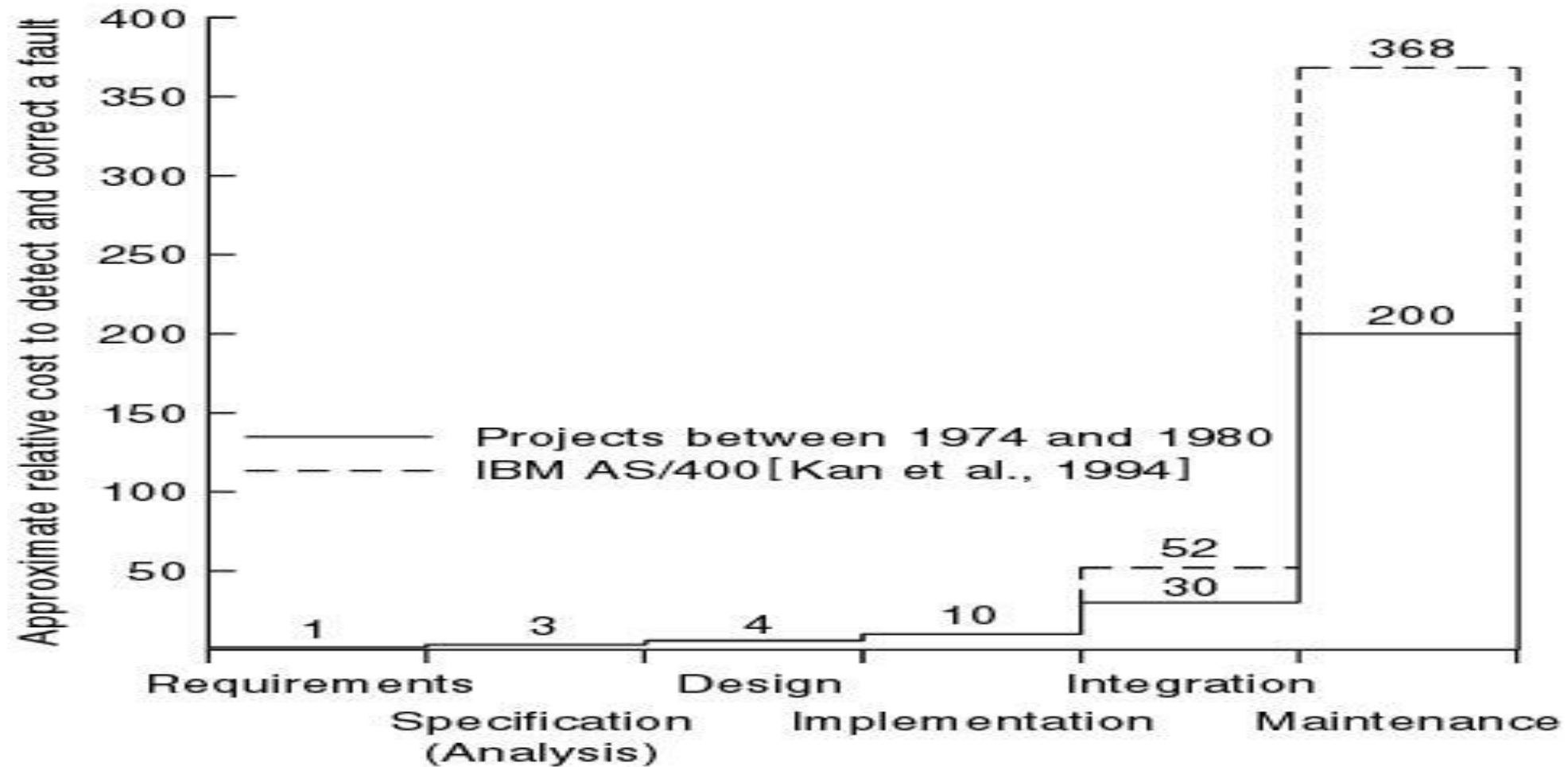
- ❑ Good software is maintained—bad software is discarded
- ❑ Different types of maintenance
  - Corrective maintenance [about 20%]
    - Modification to fix a problem
  - Enhancement [about 80%]
    - Perfective maintenance (modification to improve usability,...) [about 60%]
    - Adaptive maintenance (modification to keep up-to-date) [about 20%]
    - Preventive maintenance (modification to avoid any future error) [about 20%]

# FAULTS IN SOFTWARE DEVELOPMENT PHASES

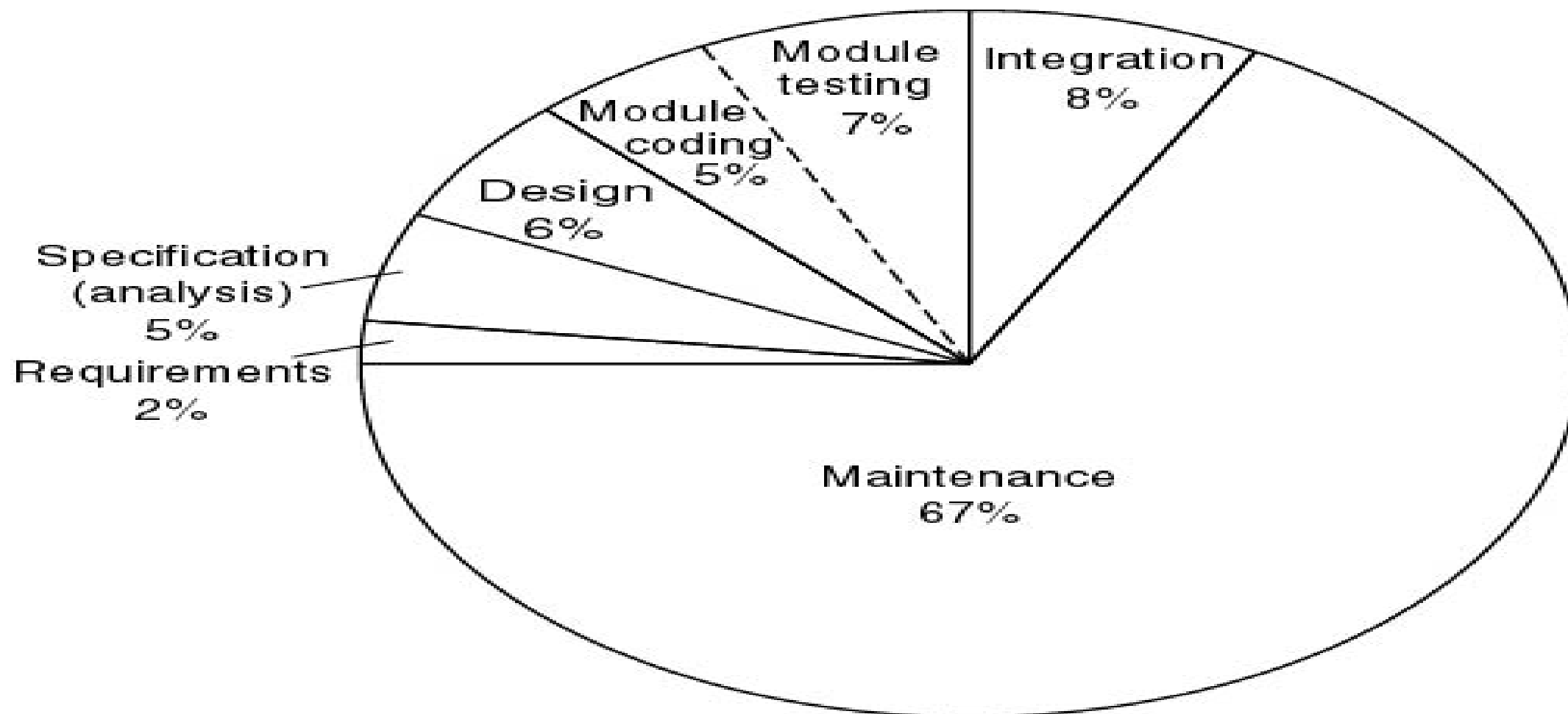
- ❑ 60 to 70 percent of faults are specification and design faults
- ❑ Data of Kelly, Sherif, and Hops [1992]
  - 1.9 faults per page of specification
  - 0.9 faults per page of design
  - 0.3 faults per page of code
- ❑ Data of Bhandari et al. [1994]
  - Faults at end of the design phase of the new version of the product
  - 13% of faults from previous version of product
  - 16% of faults in new specifications
  - 71% of faults in new design



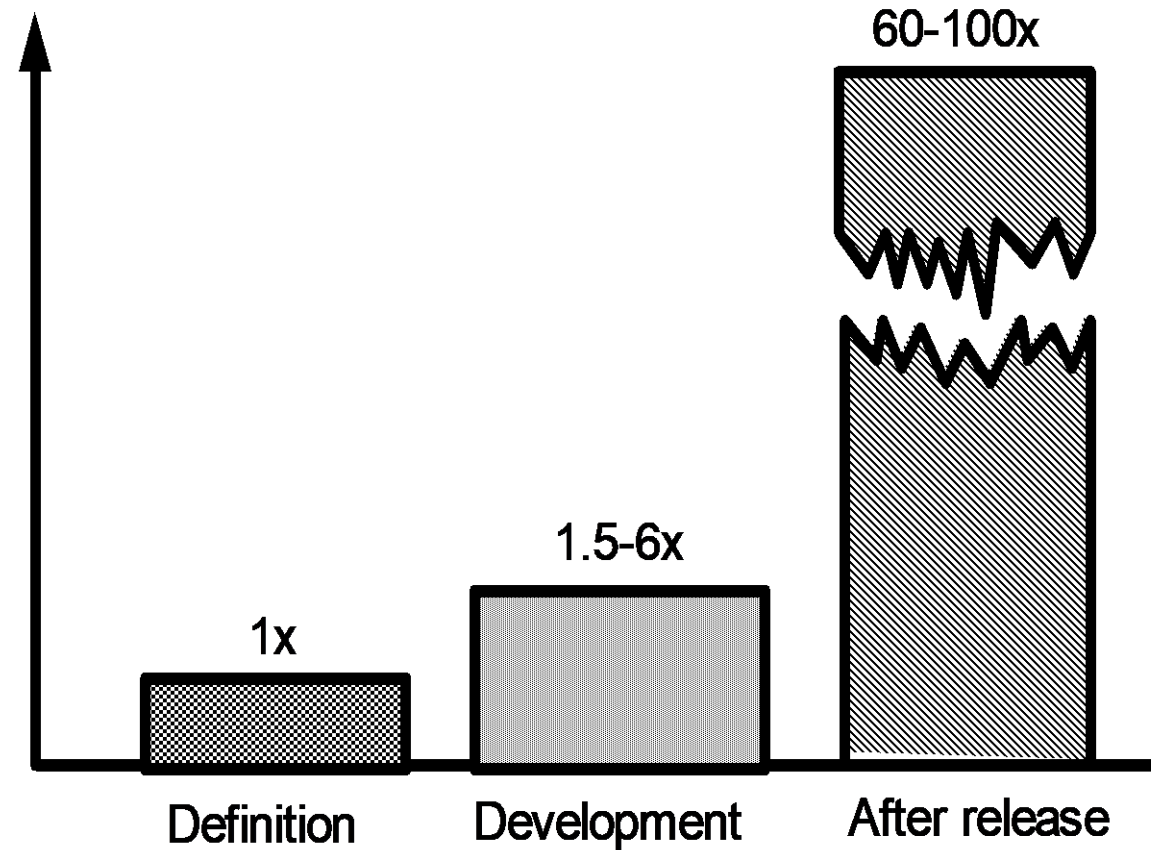
# COST OF DETECTION & CORRECTION OF A FAULT



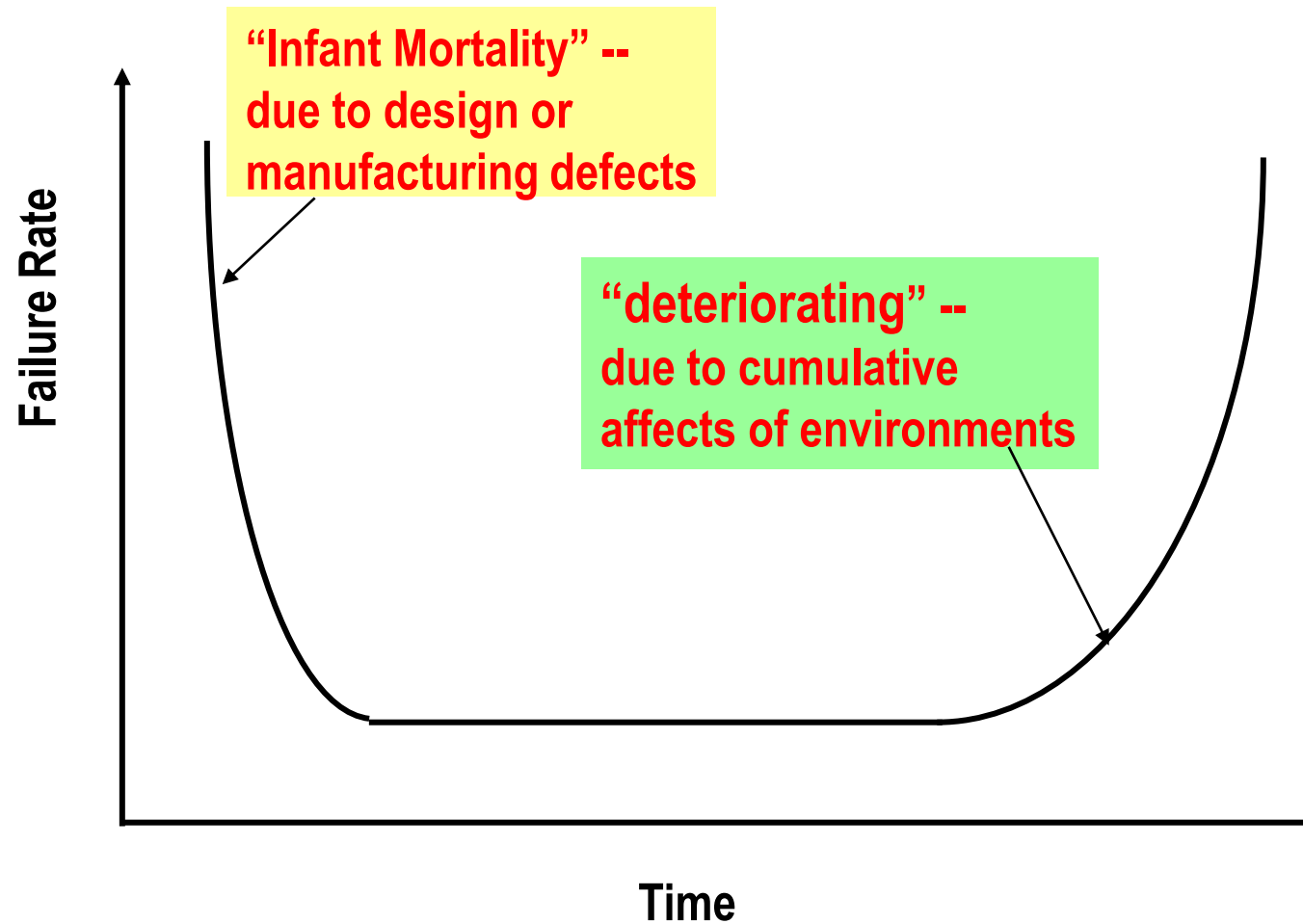
## COST OF DETECTION & CORRECTION OF A FAULT



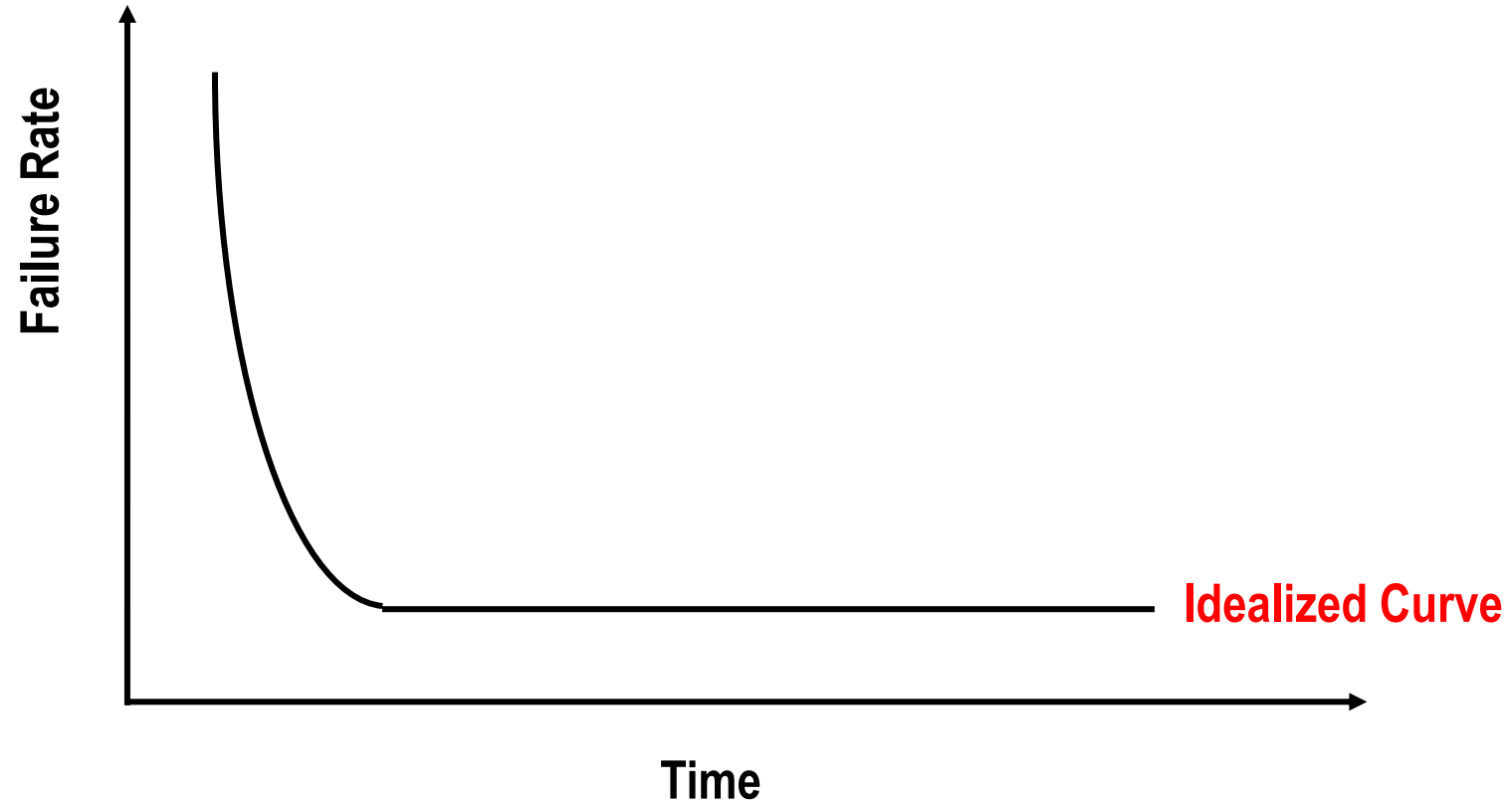
# COST OF CHANGE



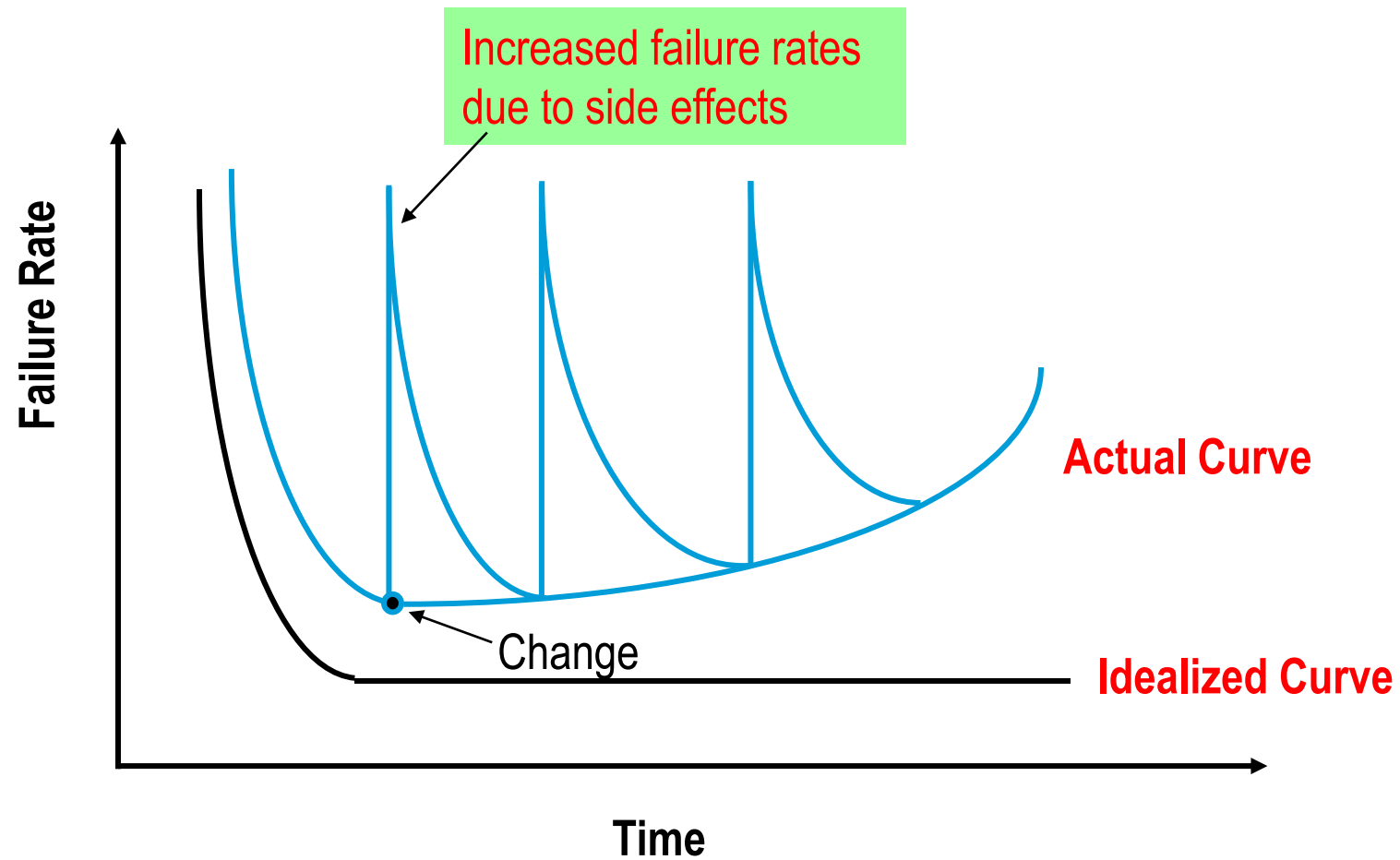
# PRODUCT BATHTUB CURVE MODEL



# SOFTWARE IDEALIZED CURVE



# SOFTWARE ACTUAL FAILURE CURVE



# WHAT IS SOFTWARE ENGINEERING?

- ❑ Technologies that make it easier, faster, and less expensive to build high-quality computer programs
- ❑ A discipline aiming to the production of fault-free software, delivered on time and within budget, that satisfies the users' needs
- ❑ **An engineering:** set of activities in software production
- ❑ The philosophy and paradigm of established engineering disciplines to solve what are termed software crisis

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

# SOFTWARE APPLICATION

- **System software** (control computer H/W such as OS)
- **Business software** (commercial application for business users, SAP, ERP)
- **Engineering and scientific software** (e.g. statistical analysis-SPSS, matlab)
- **Embedded software** (e.g. auto pilot, biometric device)
- **Personal computer software** (e.g. Microsoft Office)
- **Web-based software** (use over internet with browser, e.g. Gmail)
- **Artificial intelligence software** (interact with computer, HCI, game)



## SOFTWARE MYTHS (MANAGEMENT)

- **Myth1:** We already have a book that's **full of standards and procedures** for building s/w, won't that provide my people with everything they need to know?
- **Myth2:** My people have **state-of-the-art software development tools**, after all, we buy them the newest computers.
- **Myth3:** If we get behind schedule, we can add **more programmers** and catch up.
- **Myth4:** If I decide to outsource the software **project to a third party**, I can just relax and let that firm build it.

# SOFTWARE MYTHS (CUSTOMER)

- **Myth1:** A general statement of objectives is sufficient to begin writing programs – we can fill in the details later.
- **Myth2:** Project requirements continually change, but change can be easily accommodated because software is flexible.

# SOFTWARE MYTHS (PRACTITIONER)

- **Myth1:** Once we write the program and get it to work, our job is done.

*Fact: the sooner you begin writing code, the longer it will take you to get done.*

- **Myth2:** Until I get the program “running,” I have no way of assessing its quality.
- **Myth3:** The only deliverable work product for a successful project is the working program.
- **Myth4:** Software engineering will make us create voluminous and unnecessary documentation and will invariable slow us down.

## REFERENCES

- R.S. Pressman & Associates, Inc. (2010). *Software Engineering: A Practitioner's Approach*.
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, 17(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation. *IBM Systems Journal*, 33(1), 182-214.

COURSE NAME

SOFTWARE  
ENGINEERING

CSC 3114

(UNDERGRADUATE)

---

## CHAPTER 2

# SOFTWARE DEVELOPMENT PROCESS MODEL

---

M. MAHMUDUL HASAN

ASSISTANT PROFESSOR, CS, AIUB

<http://www.dit.hua.gr/~m.hasan>



Google Scholar

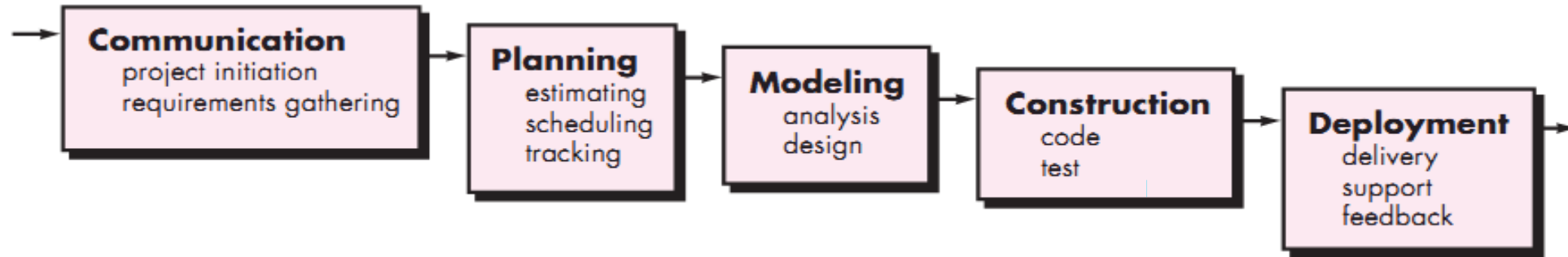


LinkedIn

# SOFTWARE PROCESS

- ❑ A structured set of activities required to develop a software system
- ❑ A software process model is an abstract representation of a process.  
It presents a description of a process from some particular perspective

# WATERFALL MODEL



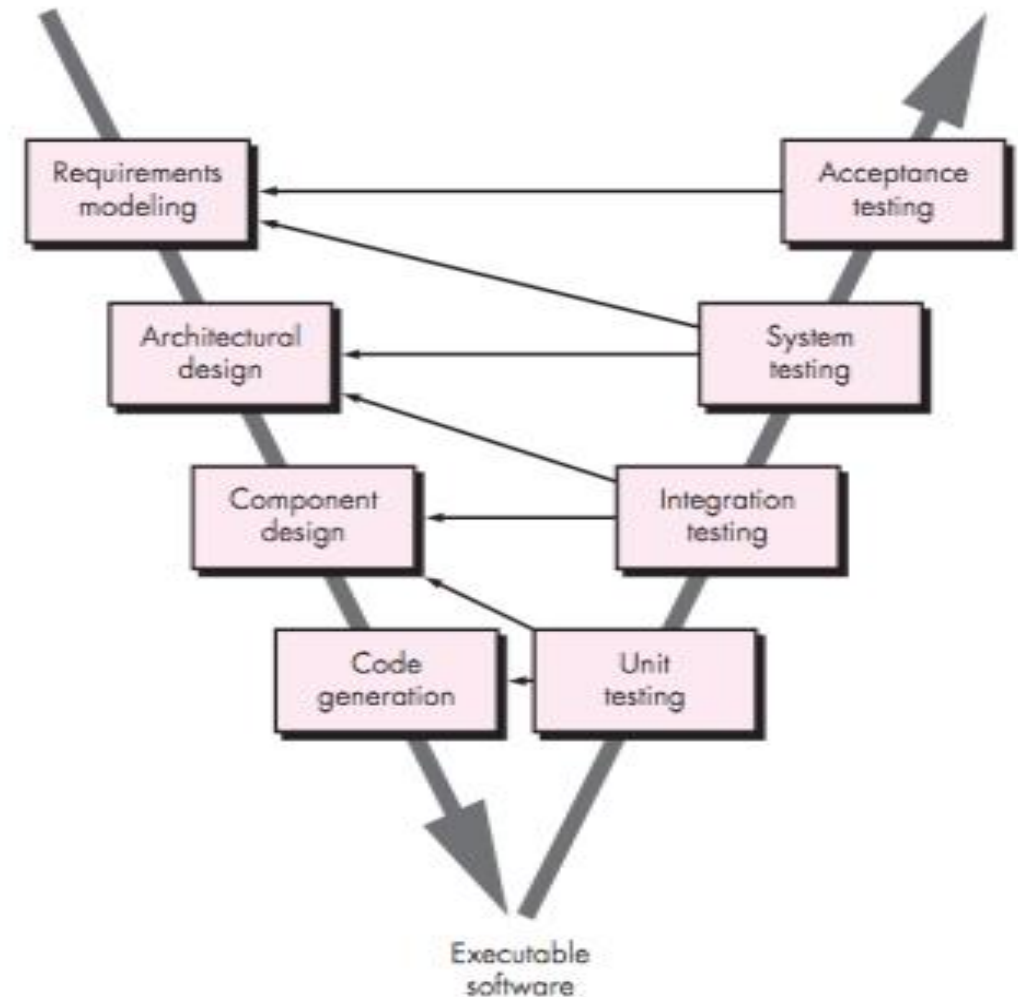
## ❑ The waterfall or linear sequential model

### Problems of Waterfall Model

- Inflexible partitioning of the project into distinct stages where next phase starts only after completion of the previous phase
- This makes it difficult to respond to changing customer requirements (no backtracking)
- Therefore, this model is only appropriate when the requirements are well-understood

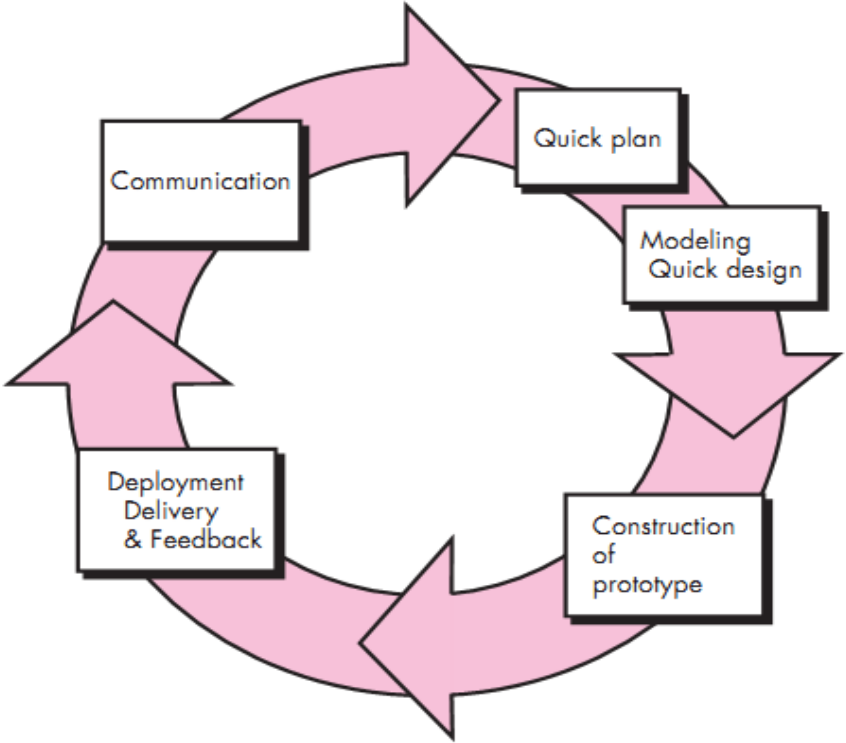
# V - MODEL

- ❑ The V-model is a SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as **Verification and Validation** model.
- ❑ V-Model is an extension of the waterfall model and is based on association of a **testing phase** for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase.
- ❑ This is a highly disciplined model and next phase starts only after completion of the previous phase.



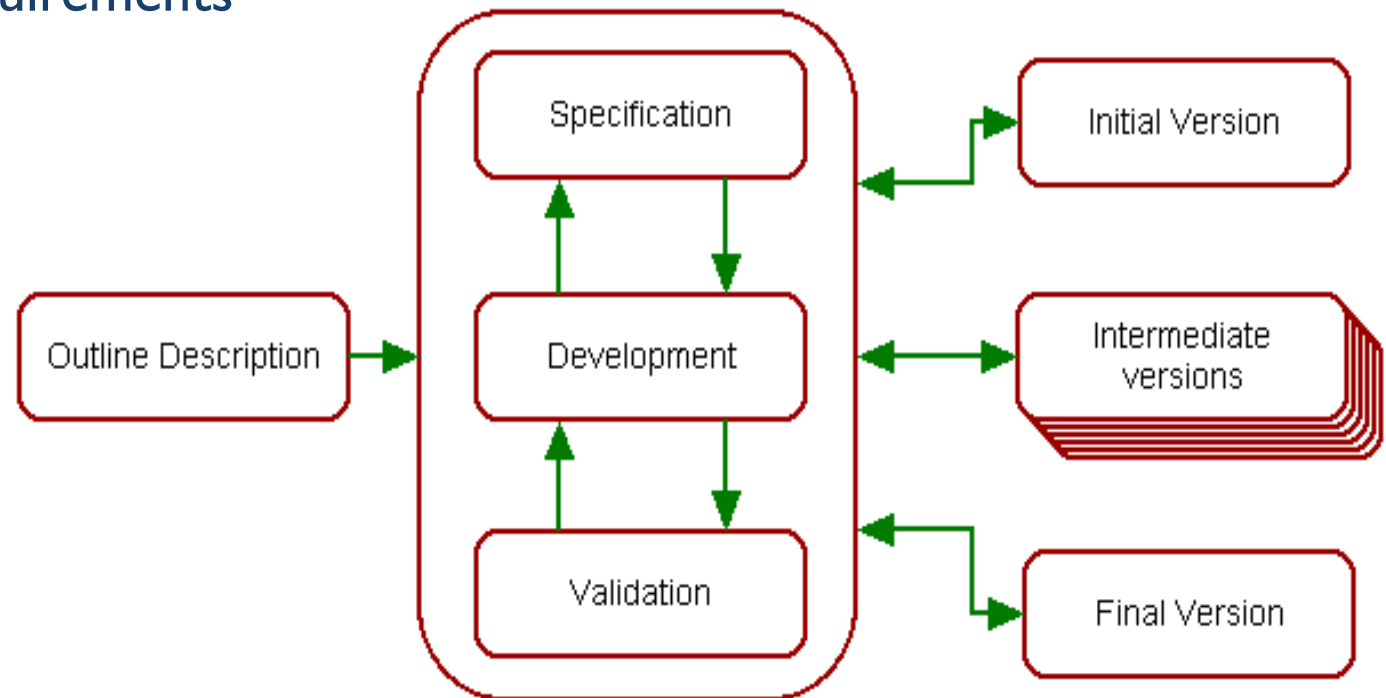


# PROTOTYPING MODEL

- ❑ Requirements are not clear and prototype serves as a mechanism for identifying software requirements
  - ❑ Iteration occurs as the prototype is tuned to satisfy the needs of the customer
- 
- ❑ System requirements ALWAYS evolve in the course of a project, so process iteration is useful where earlier stages are reworked is always part of the process for large systems

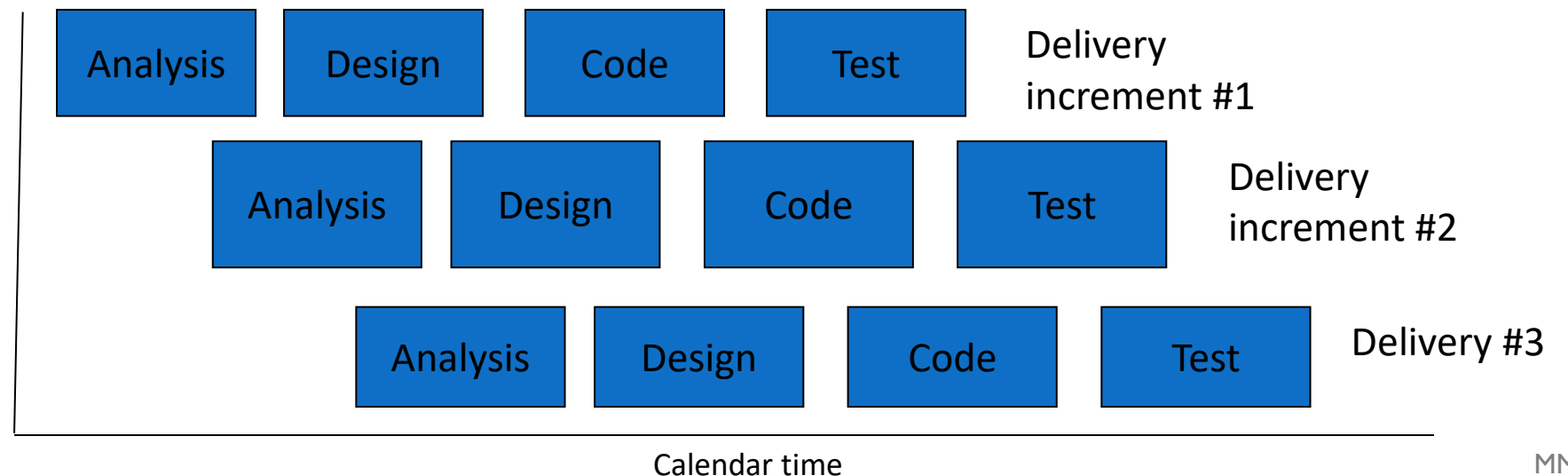
# EVOLUTIONARY DEVELOPMENT

- ❑ **Exploratory development:** Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements
- ❑ **Throw-away prototyping:** Objective is to understand the system requirements. Should start with poorly understood requirements



# INCREMENTAL DEVELOPMENT

- ❑ Rather than deliver the system as a single delivery, the development and **delivery is broken down into increments** with each increment delivering part of the required functionality (SPIRAL).  
The requirements are relatively certain but there are many complexities that leads to frequent changes.
- ❑ User requirements are prioritised and the **highest priority requirements** are included in early increments
- ❑ Once the development of an increment is started, the **requirements are frozen** though requirements for later increments can continue to evolve



# INCREMENTAL DEVELOPMENT

## ❑ Advantages of Incremental Development

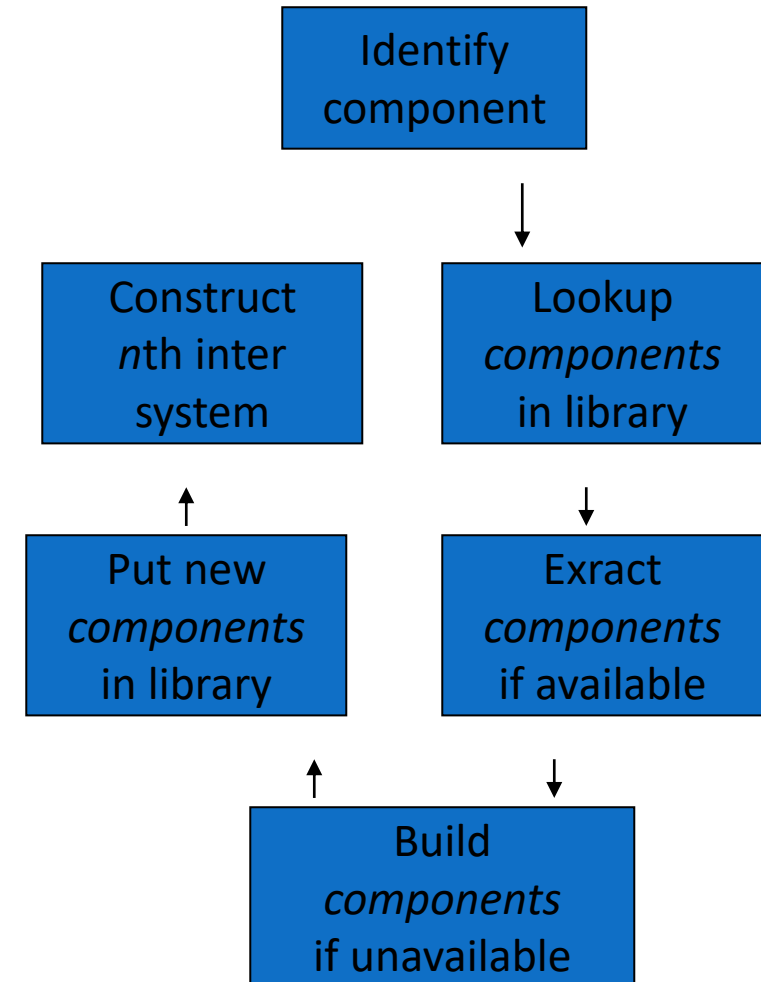
- Customer value can be delivered with each increment so system functionality is available earlier
- Deliver the core product first
- Early increments act as a prototype to help elicit requirements for later increments
- Lower risk of overall project failure
- The highest priority system services tend to receive the most testing

# RAPID APPLICATION DEVELOPMENT (RAD)

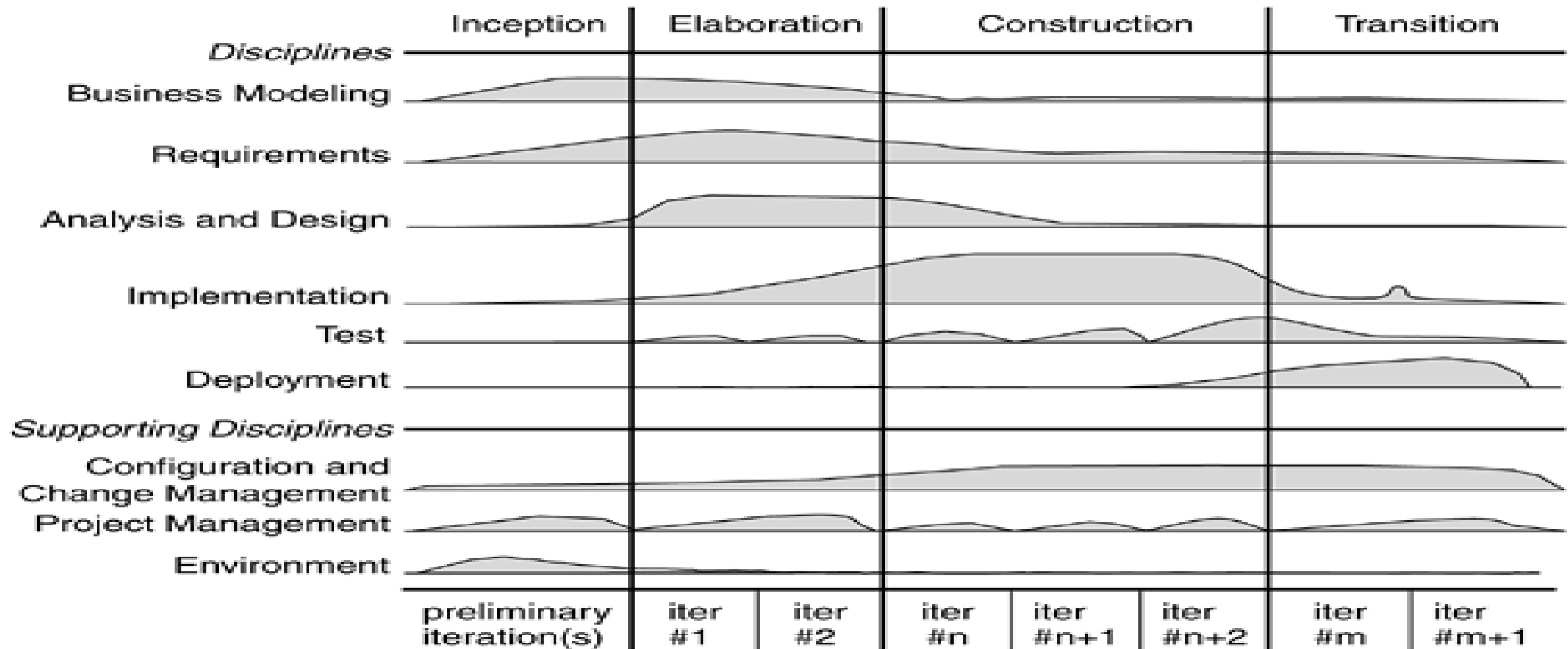
- ❑ It is a type of incremental model. The developments are time boxed, delivered and then assembled into a working prototype
- ❑ In RAD model the components or functions are **developed in parallel** as if they were mini projects (frozen requirements in each increments)
- ❑ This can quickly give the customer something to see and use and to provide feedback
- ❑ Delivers a fully functional system in **90 days**, give or take 30 days
- ❑ Phases of RAD are:
  - Requirements Planning
  - User Design (user interact with the system analysts)
  - Construction (program and application development)
  - Cutover (testing, changeover to new system, user training)

# COMPONENT BASED DEVELOPMENT MODEL

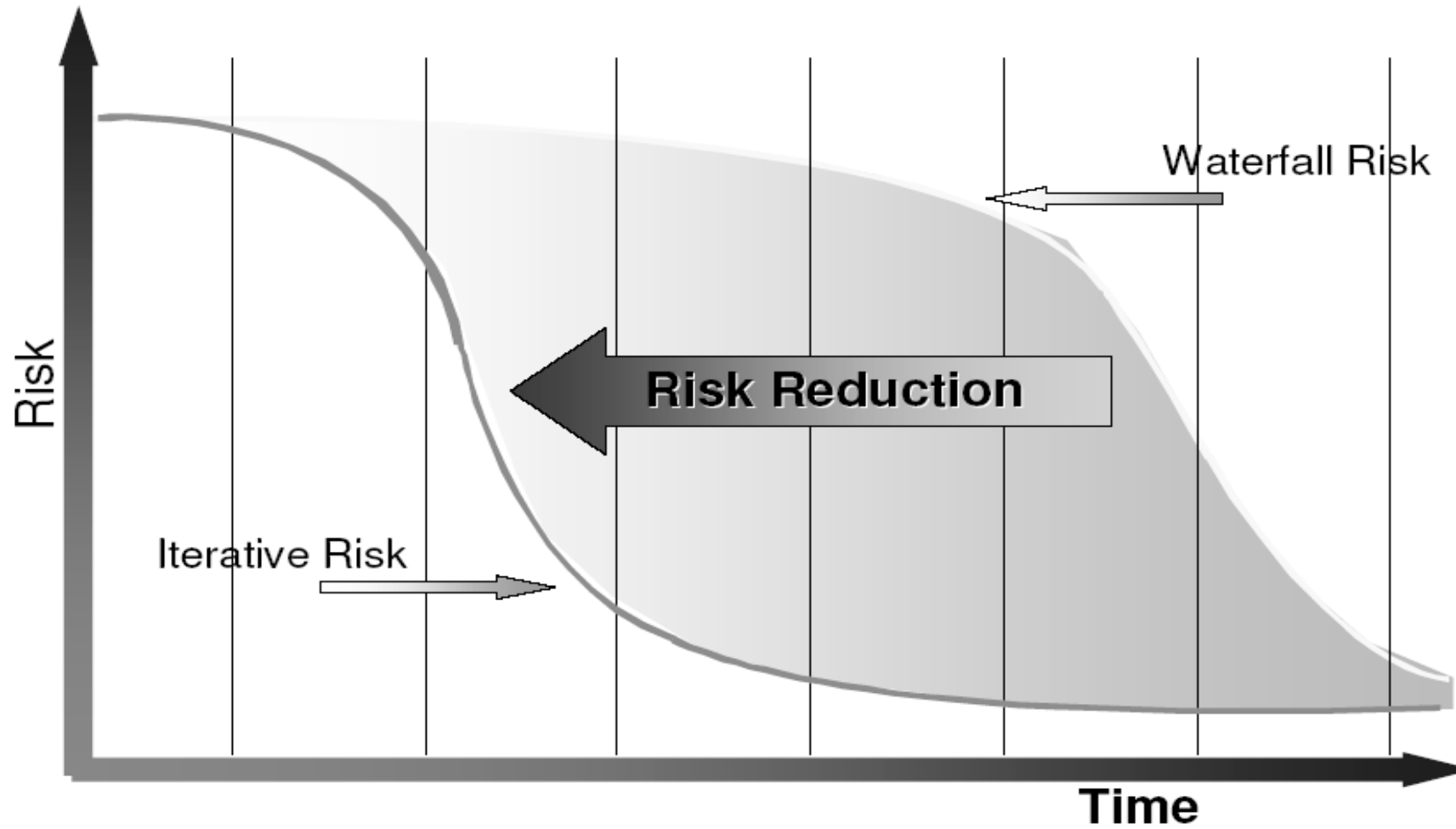
- ❑ Suitable for re-usable object oriented classes
- ❑ Apply characteristics of spiral development



# RATIONAL UNIFIED PROCESS (RUP)

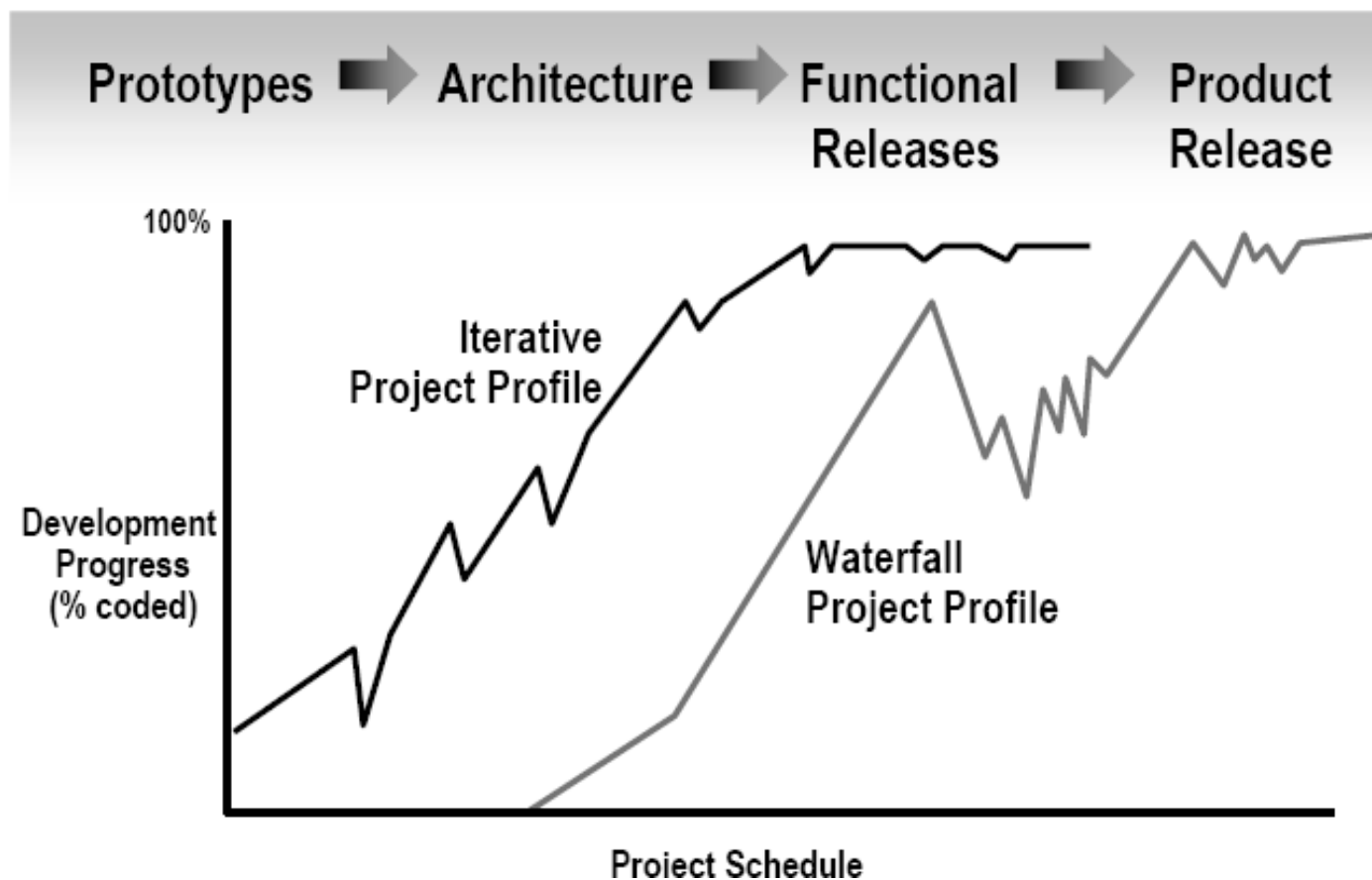


# RISK PROFILE





# REDUCE SCRAP/REWORK: USE AN ITERATIVE PROCESS



## □ Iterative Development

- Products are visible at an early stages of development
- Low probability of rework in case of defects in the deliverable product

## REFERENCES

- R.S. Pressman & Associates, Inc. (2010). *Software Engineering: A Practitioner's Approach*.
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, 17(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation. *IBM Systems Journal*, 33(1), 182-214.

COURSE NAME

SOFTWARE  
ENGINEERING

CSC 3114

(UNDERGRADUATE)

---

## CHAPTER 3

# AGILE SOFTWARE DEVELOPMENT

---

M. MAHMUDUL HASAN

ASSISTANT PROFESSOR, CS, AIUB

<http://www.dit.hua.gr/~m.hasan>



Google Scholar



LinkedIn

# AGILE DEVELOPMENT

“Plan-driven methods work best when developers can determine the requirements in advance ... and when the requirements remain relatively stable, with change rates on the order of one percent per month.”~ *Barry Boehm*

□ Agility is the ability to create and respond to change in order to profit in a turbulent business environment

□ Companies need to

- innovate better and faster operations
- respond quickly to
  - competitive initiatives
  - new technology
  - customer's requirements

The Agile model is an iterative and incremental approach to software development.

One of the key advantages of the Agile model is its ability to quickly adapt to changing requirements and customer needs

# AGILE MODEL

- ❑ Subset of iterative and evolutionary methods

## Iterative Products

- Each iteration is a **self-contained, mini-project** with activities that span requirements analysis, design, implementation, and test
- Leads to an iteration release (which may be only an **internal release**) that integrates all software across the team and is a growing and evolving subset of the final system
- The purpose of having short iterations is so that feedback from iterations N and earlier, and any other new information, can **lead to refinement** and requirements adaptation for iteration  $N + 1$

# AGILE METHODS VS. PAST ITERATIVE METHODS

- A key difference between agile methods and past iterative methods is the length of each iteration
  - In the past, iterations might have been three or six months long
  - In agile methods, iteration lengths vary between one to four weeks, and intentionally do not exceed 30 days
  - Research has shown that shorter iterations have lower complexity and risk, better feedback, and higher productivity and success rates

## TIMEBOX & SCOPE

- ❑ The pre-determined iteration length serves as a timebox for the team.
- ❑ Scope (set of tasks) is chosen for each iteration to fill the iteration length.
- ❑ Rather than increase the iteration length to fit the chosen scope, the scope is reduced to fit the iteration length.

## AGILE PROPONENTS BELIEVE

- ❑ Current software development processes are too heavyweight or cumbersome
  - Too many things are done that are not directly related to software product being produced
- ❑ Current software development is too rigid
  - Difficulty with incomplete or changing requirements
- ❑ Short development cycle is needed
- ❑ More active customer involvement needed



# WHAT IS AN AGILE METHOD?

- ❑ Agile methods are considered
  - Lightweight (do not concentrate on the whole s/w development at once)
  - People-based rather than Plan-based
- ❑ Several agile methods
  - No single agile method
  - Different agile methods can be combined in s/w development (Hybrid)
- ❑ No single definition. Agile Manifesto closest to a definition
  - Set of principles
  - Developed by Agile Alliance

# AGILE VALUE STATEMENT

- ❑ In 2001, Kent Beck and 16 other noted software developers, writers, and consultants (known as Agile Alliance) signed a manifesto as following:

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:”

- ***individuals and interactions*** over processes and tools
- ***working software*** over comprehensive documentation
- ***customer collaboration*** over contract negotiation
- ***responding to change*** over following a plan

# AGILE VS. PLAN DRIVEN PROCESS

Agile Process	Plan Driven Process
Small products and teams; scalability limited	Large products and teams; hard to scale down
Inappropriate for safety-critical products because of frequent changes	Handles highly critical products
Good for dynamic, but expensive for stable environments.	Good for stable, but expensive for dynamic environments
Require experienced Agile personnel throughout	Require experienced personnel only at start if stable environment
Personnel succeed on freedom and chaos	Personnel succeed on structure and order

# AGILE ASSUMPTION

- ❑ It is difficult to predict in advance which software
  - requirements will persist and which will change
  - It is equally difficult to predict how customer priorities will change as the project proceeds
- ❑ Design and construction are interleaved in many types of software. That is, both activities should be performed tightly so that design models are proven as they are created. It is difficult to predict how much design is necessary before construction is used to prove the design.
- ❑ Analysis, design, construction, and testing are not as predictable (from a planning point of view) as we might like.

# AGILE MANIFESTO (POLICY)

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
2. Welcome changing requirements, even late in development. Agile process harness (control) change for the customer's competitive advantage
3. Deliver working software frequently with a preference to the shorter timescale
4. Business people and developers must work together daily throughout the project
5. Build projects around motivated individuals. Give them the environment and support their need, and trust them to get the job done
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

# AGILE MANIFESTO (POLICY)

7. Working software is the primary measure of progress
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace persistently
9. Continuous attention to technical excellence and good design enhances agility
10. Simplicity – use simple approaches to make changes easier
11. The best architectures, requirements, and designs emerge from self-organizing teams (iterative development rather than defined plans)
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

# HUMAN FACTORS IN AGILE DEVELOPMENT

- ❑ Competence/skill/capability
- ❑ Common focus
- ❑ Collaboration
- ❑ Decision-making ability
- ❑ Fuzzy (vague) problem-solving ability
- ❑ Mutual trust and respect
- ❑ Self-organization

# AGILE METHODS

- ❑ Extreme Programming (XP)
- ❑ Scrum
- ❑ Dynamic Systems Development Method (DSDM)
- ❑ Feature-Driven Development (FDD)
- ❑ Crystal Methods
- ❑ Lean Development (LD)
- ❑ Adaptive Software Development (ASD)



## REFERENCES

- R.S. Pressman & Associates, Inc. (2010). *Software Engineering: A Practitioner's Approach*.
- Kelly, J. C., Sherif, J. S., & Hops, J. (1992). An analysis of defect densities found during software inspections. *Journal of Systems and Software*, 17(2), 111-117.
- Bhandari, I., Halliday, M. J., Chaar, J., Chillarege, R., Jones, K., Atkinson, J. S., & Yonezawa, M. (1994). In-process improvement through defect data interpretation. *IBM Systems Journal*, 33(1), 182-214.