SLIIT
Discover Your Future

SOFTWARE PROCESS MODELING

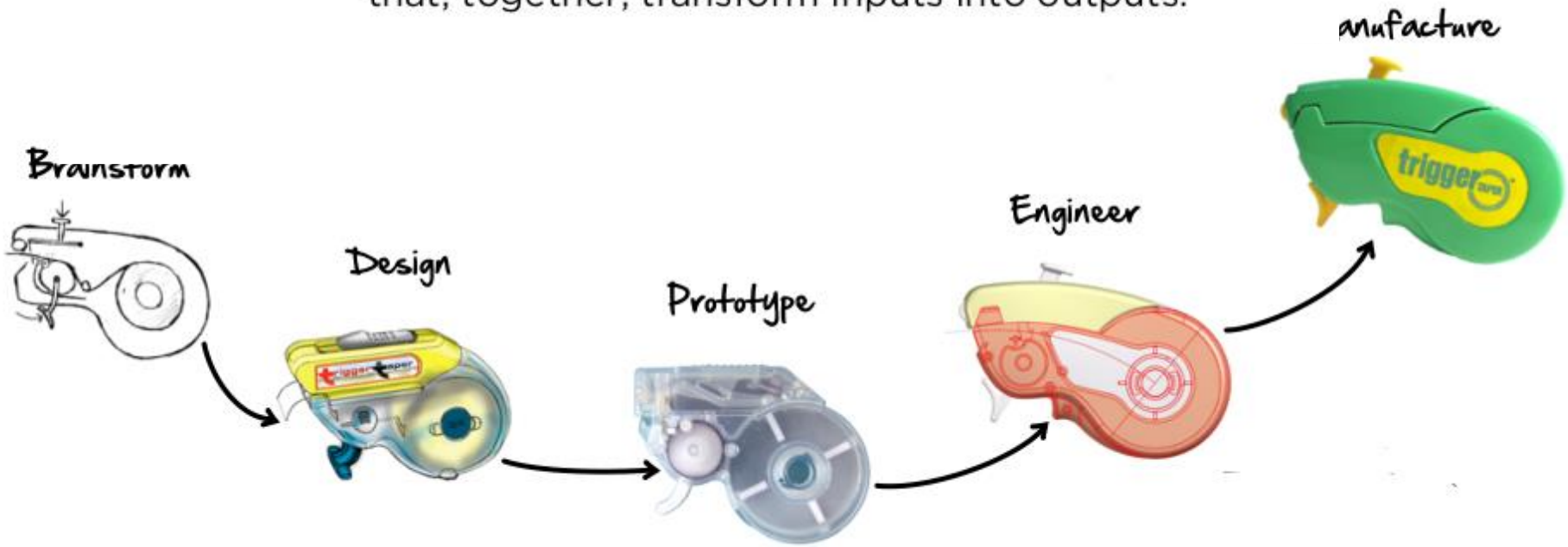
Software Development Life Cycles

Session Outcomes

- What is a
  - Software Process
  - SDLC
  - Life Cycle Model
- Life Cycle Models
  - Traditional Approaches
  - Modern Approaches
- Comparison and Selection
- Activities

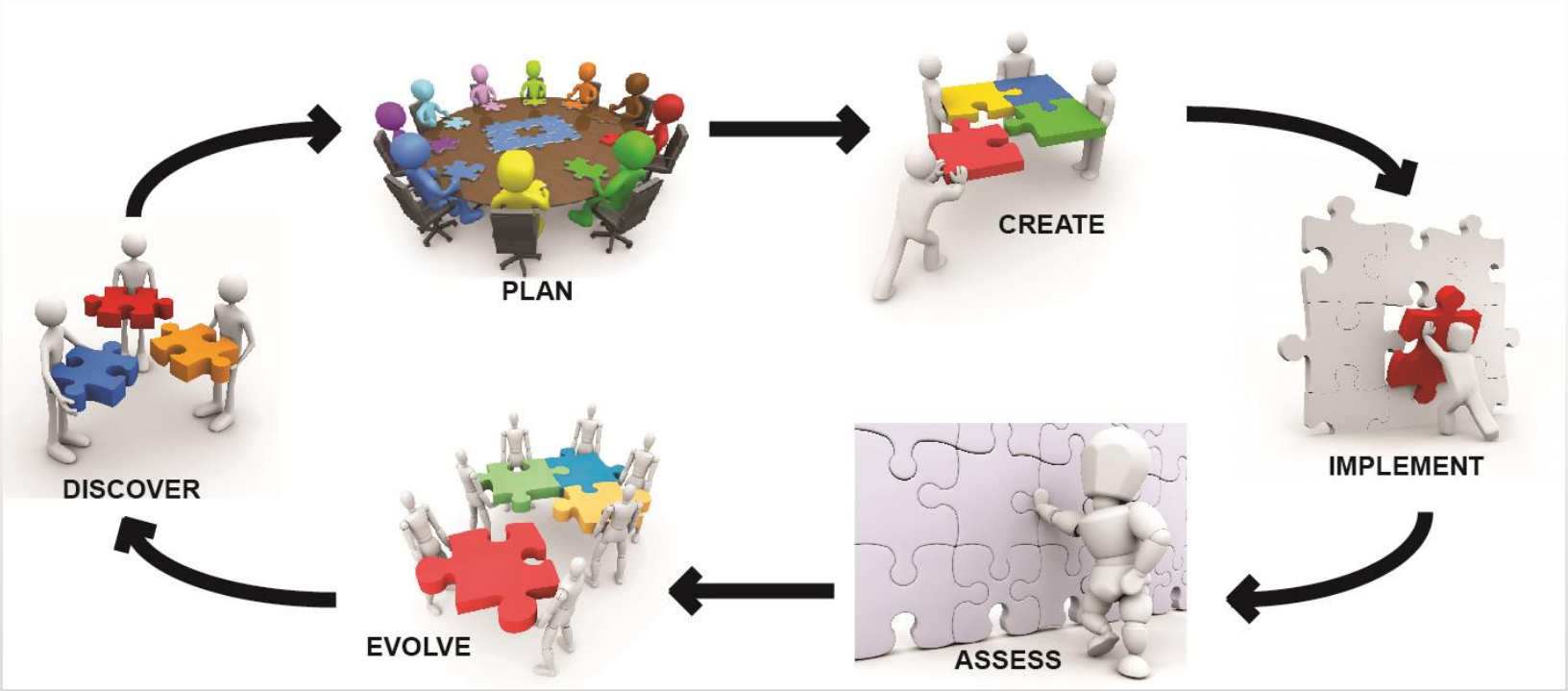
What is an Engineering Process?

In engineering a process is a set of interrelated tasks that, together, transform inputs into outputs.



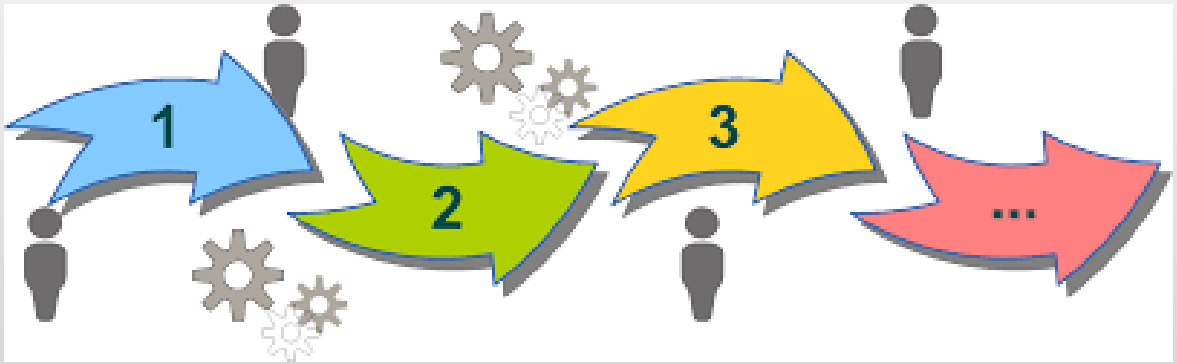
What is a Software Development Process?

- A set of Activities and Associated Results that produce a Software.



Fundamental Process Activities

1. Software Specification
2. Software Development
3. Software Validation
4. Software Evolution



1. Software Specification

- Specification involves clearly documenting the expectations on the system to be built
- A **software** requirements **specification** (SRS) is a description of a **software** system to be developed.
- It lays out requirements, and may include written and diagrammatic description of the services that the future system must provide.

2. Software Development

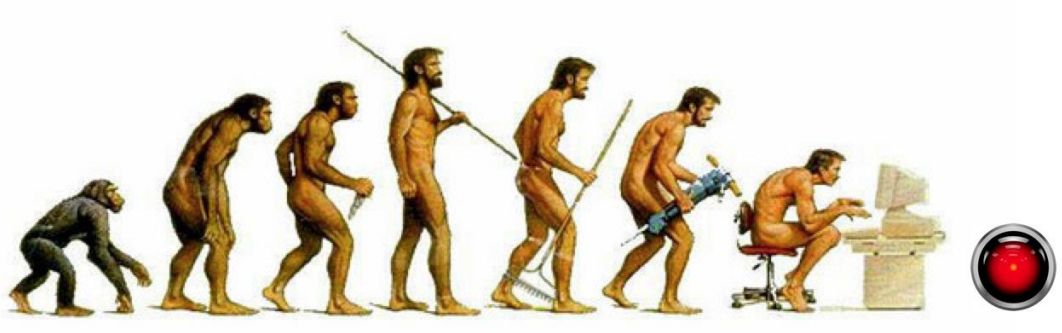
- Software development involves designing and implementing the system according to the software specification

3. Software Validation

- Software validation involves checking and verifying whether the system fulfills the requirements.
- Different types of tests can be carried out.

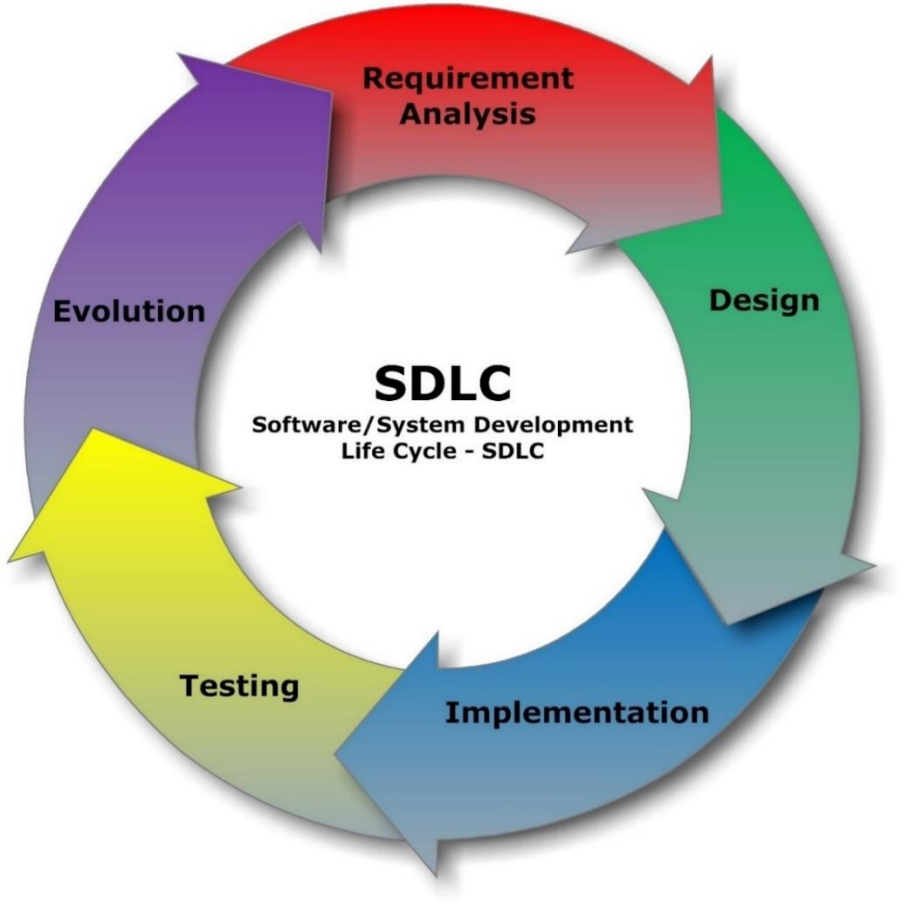
4. Software Evolution

- Software needs to be changed and upgraded with time.
- Such changes are carried out as maintenance activities



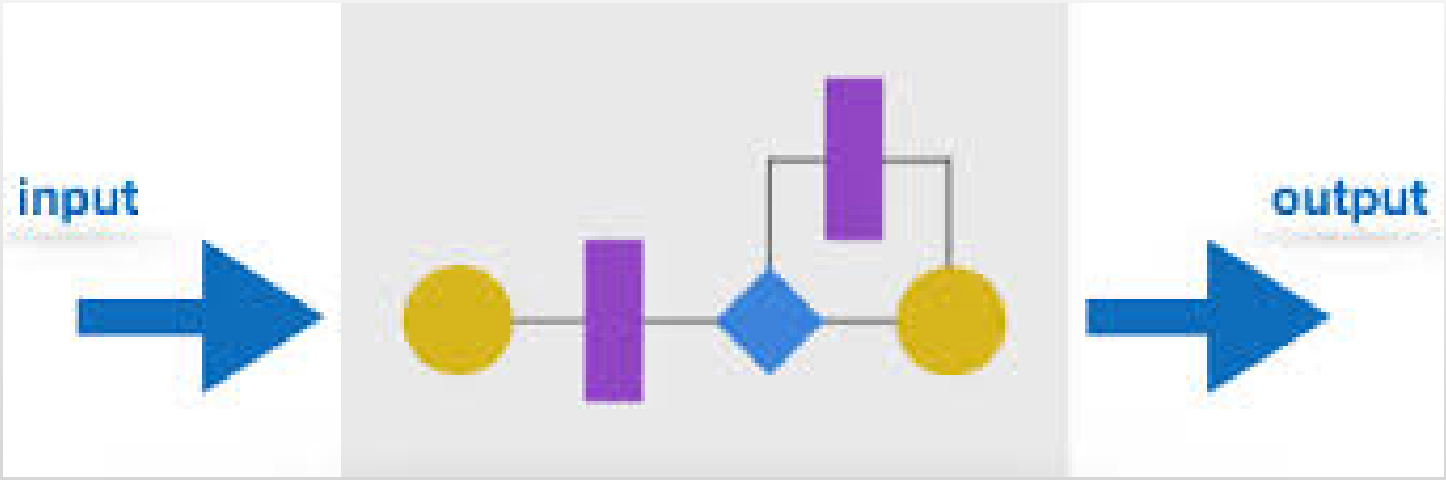
What is a SDLC?

- The Software Development Life Cycle (SDLC) is a framework that defines activities performed throughout the software development process



Software Process Model (Life Cycle Model)

- A Software Process Model is a simplified description of a Software Process and represents one view of that process.
- In different process models, the order in which various phases occur may differ.



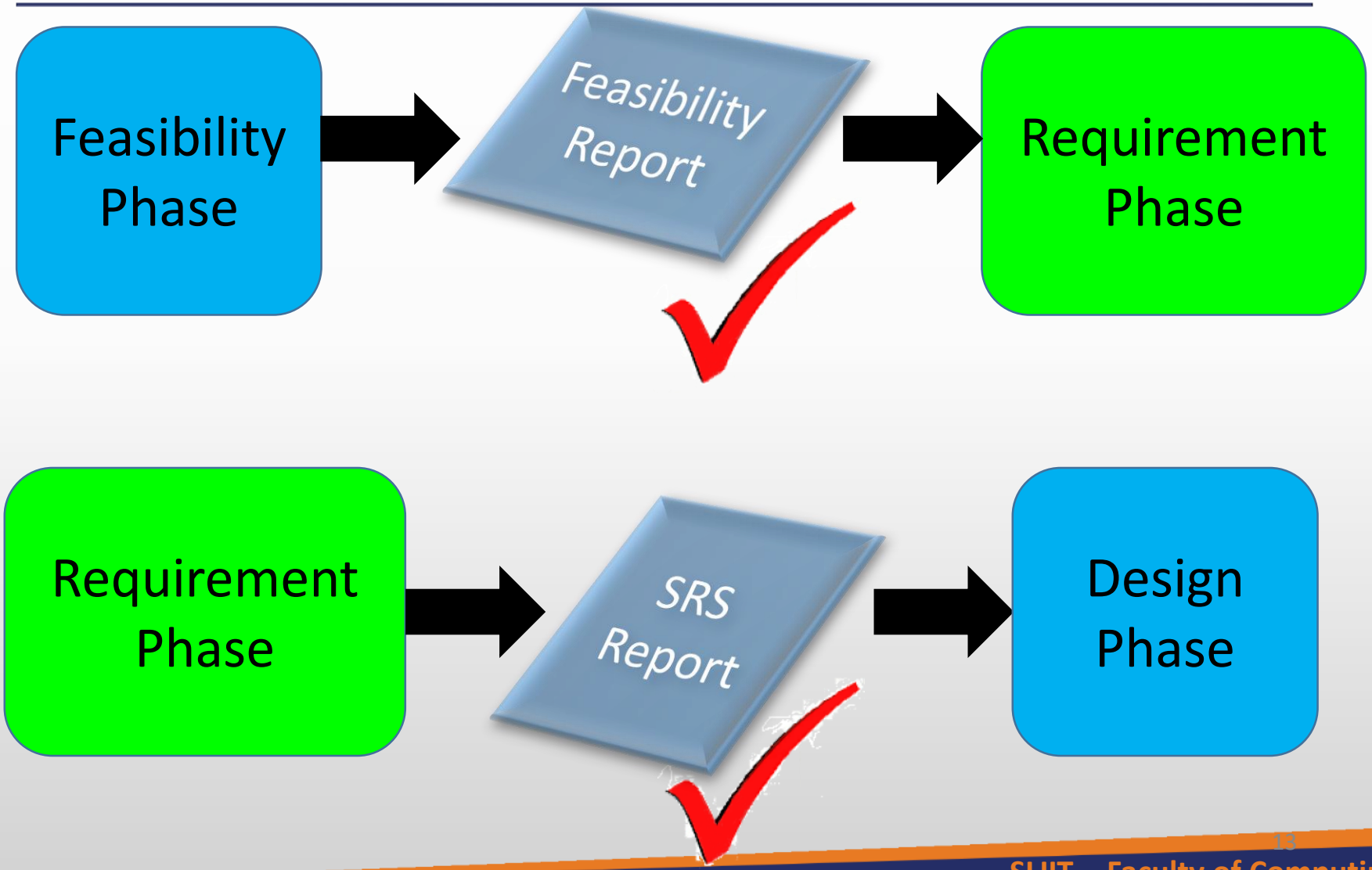
Life Cycle (Process Model)

- A life cycle is a series of **phases (steps)** that software passes through during its lifetime.

- A life cycle (process) model defines entry and exit criteria for every phase:
  - a phase begins when all of its **entry criteria** are met;
  - a phase ends when all of its **exit criteria** are met;
  - the entry and exit criteria make it easier to monitor the progress of a project.

- Example exit criteria for the coding phase:
  - each module has been tested and documented.

Exit / Entry Criteria E.g.



Life Cycle (Process Model)

- A software life cycle (process) model:
  - is a descriptive and diagrammatic model of the life cycle of a software product;
  - identifies all the activities and phases necessary for software development;
  - establishes a precedence ordering among the different activities.

- Life cycle models encourage systematic and disciplined software development.

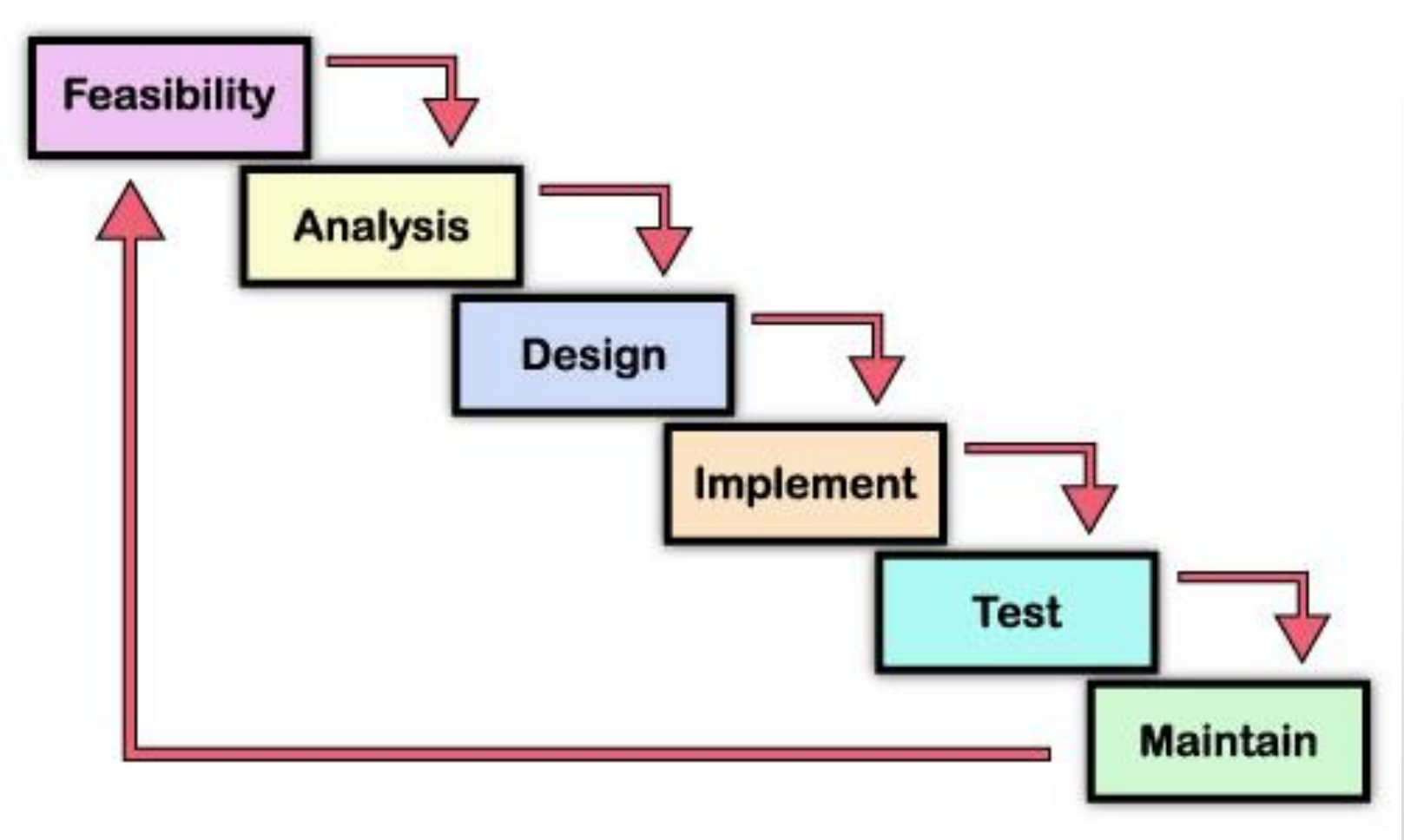
Life Cycle Models

- Traditional Approaches
  1. Waterfall Model
  2. Incremental Model
  3. Prototyping Model
  4. Spiral Model
  5. Unified Process
- Modern Approaches
  - Agile Methods (XP, Scrum)
  - Secure Software Development

Life Cycle Models

1
Waterfall Model

Waterfall Model



Waterfall (Linear) model

- Waterfall model is the most well known software lifecycle development model.
- It is very simple to understand and use.
- Each phase begins only after the previous phase is over.
- This model specifies what the system is supposed to do (i.e. define the requirements) before building the system (i.e. designing)

BRAND CAMP

by Tom Fishburne

THE NEW PRODUCT WATERFALL

HOW DO WE CHART OUR ENTIRE COURSE IF WE DON'T KNOW WHAT'S AHEAD?

PLAN

WHATEVER HAPPENS, JUST KEEP PADDLING!

BUILD

I WISH WE'D DESIGNED FOR THIS SCENARIO UPFRONT

TEST

PATCH IT AS BEST WE CAN. NO TIME TO CHANGE COURSE NOW

LAUNCH

1. Feasibility Study

- This is the first phase of any SDLC model.
- The project objective is determined during this phase.
- The client and company developing the software decide if they should ;
    - Keep the existing system as is, or
    - Build a new software

Why do a feasibility study?

- To provide management with enough information to know:
  - Whether the project can be done
  - Whether the final product will benefit its users
  - What the alternative solutions are
  - Whether there is a preferred alternative

Feasibility Study - Steps

1. Gain the basic understanding of the problem.
2. Formulate alternative solution strategies.
3. Evaluate solution strategies in terms of:
    - Technical feasibility
    - Economical feasibility
    - Operational feasibility
    - Schedule feasibility

4. Prepare a report with different recommendations for the system (personnel, costs, schedule and target dates) and submit for management approval.

Feasibility Study

- Technical feasibility
  - Is the project possible with current technology?
    - Technology available in-house?
    - Will it be compatible with other systems?
- Schedule feasibility
  - Is it possible to build a solution in time ?
    - What are the consequences of delay?
    - Any constraints on the schedule?
    - Can these constraints be met?
- Economic feasibility
  - Is the project possible, given resource constraints?
    - What are the development and operational costs?
    - What are the benefits? (tangible, intangible)
    - Are the benefits worth the costs?
- Operational feasibility
  - If the system is developed, will it be used?
  - Human and social issues…
    - Potential objections? resistance?
    - Organizational conflicts and policies?
    - Social acceptability?
    - legal aspects and government regulations?

Feasibility Study

- A Cost/Benefit Analysis can be used to decide which solutions are feasible

Benefits

- Tangible Benefits
  - increased sales
  - cost/error reductions
  - increased throughput/efficiency
  - increased margin on sales
  - more effective use of staff time
- Intangible benefits
  - increased flexibility of operation
  - higher quality products/services
  - better customer relations
  - improved staff morale

Costs

- Development costs
  - Development and purchasing costs:
    - Cost of development team, Consultant fees
    - software , hardware
    - facilities (site, communications.)
  - Installation and conversion costs:
    - installing the system,
    - training,
    - file conversion
- Operational costs (on-going)
  - System Maintenance:
    - hardware (repairs, lease,...)
    - software (licenses and contracts),
  - Personnel:
    - For operation
    - For support
    - On-going training

Example – Library System

- What are the feasibility criteria for a ‘Library System’?

2. The Requirements Phase

- Aim: to understand the customer's requirements:
  - i.e. the PROBLEM!
- A customer may be a single person, a group, a department, or an entire organization:
  - often with many employees (potential users).
- This phase involves two distinct activities:
  1. **Requirements Gathering**
  2. **Requirements Analysis**
  3. **Requirements Specification**

Requirement Gathering

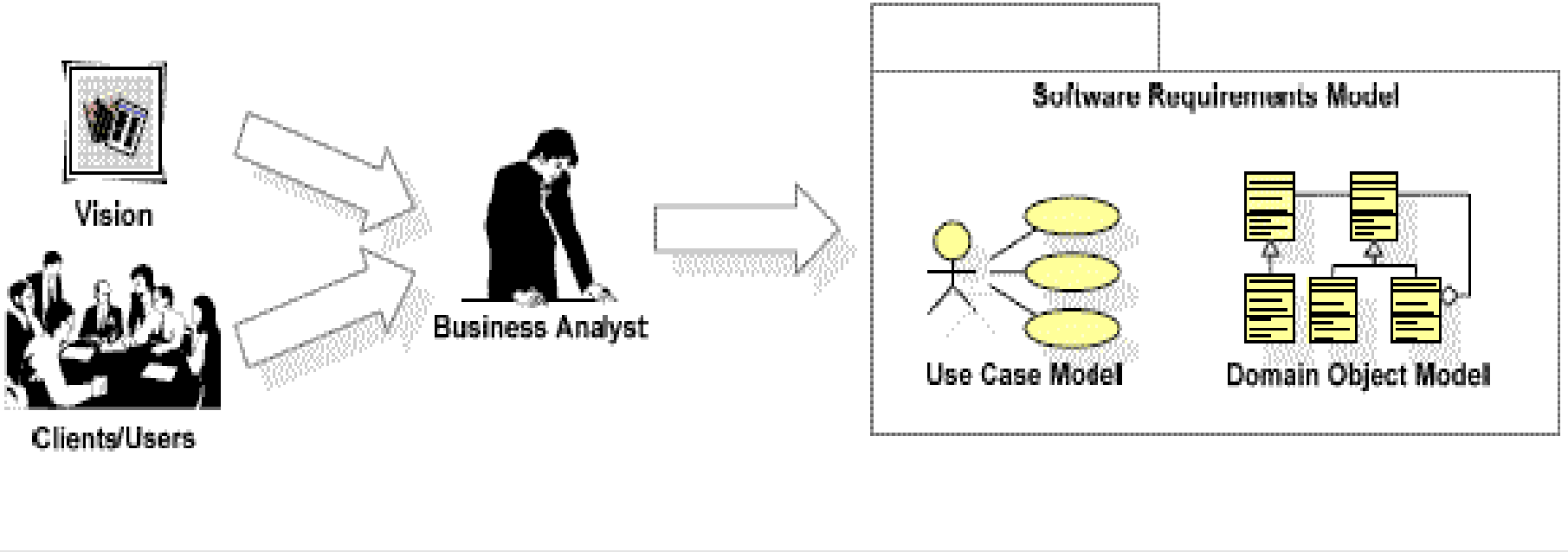
- The goal of this phase is for the stakeholders to find out the 'what to be done'.
- Questions answered during this phase include:
    - Who will use the system?
    - How will they use the system?
    - What will the input be for the system?
    - What will the output be for the system?
- Requirement Gathering involve collecting information through meetings, interviews and discussions

Requirements Analysis

- **Aim:** To understand exactly what the customer needs.. which may not be what they ask for:
  - data to be **input** to the system;
  - **processing** to be performed on these data;
  - data to be **output** from the system;
  - **characteristics** of the system as a whole;
  - **constraints** on the system/project.

- WHAT, not HOW!

Requirements Analysis



Requirements Specification

- Requirements are documented in a Software Requirements Specification (SRS).
- The SRS forms the basis of a legal contract with the customer:
  - Customer approval of the SRS document typically marks the end of the requirements phase.

- Software Engineers who specialize in requirements gathering, analysis, and specification are called (Systems/ Business / Requirement) Analysts.

3. Design

- Architects and Designers craft a high-level and low level design of the software.
  - Architectural Design
  - Low level Design
- Decisions are made about hardware, software and the system architecture.
- A design specification document (DSD) records this information.

4. Development

- On receiving system design documents, the work is divided in modules.
- A set of developers code the software as per the established design specification, using a chosen programming language
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process

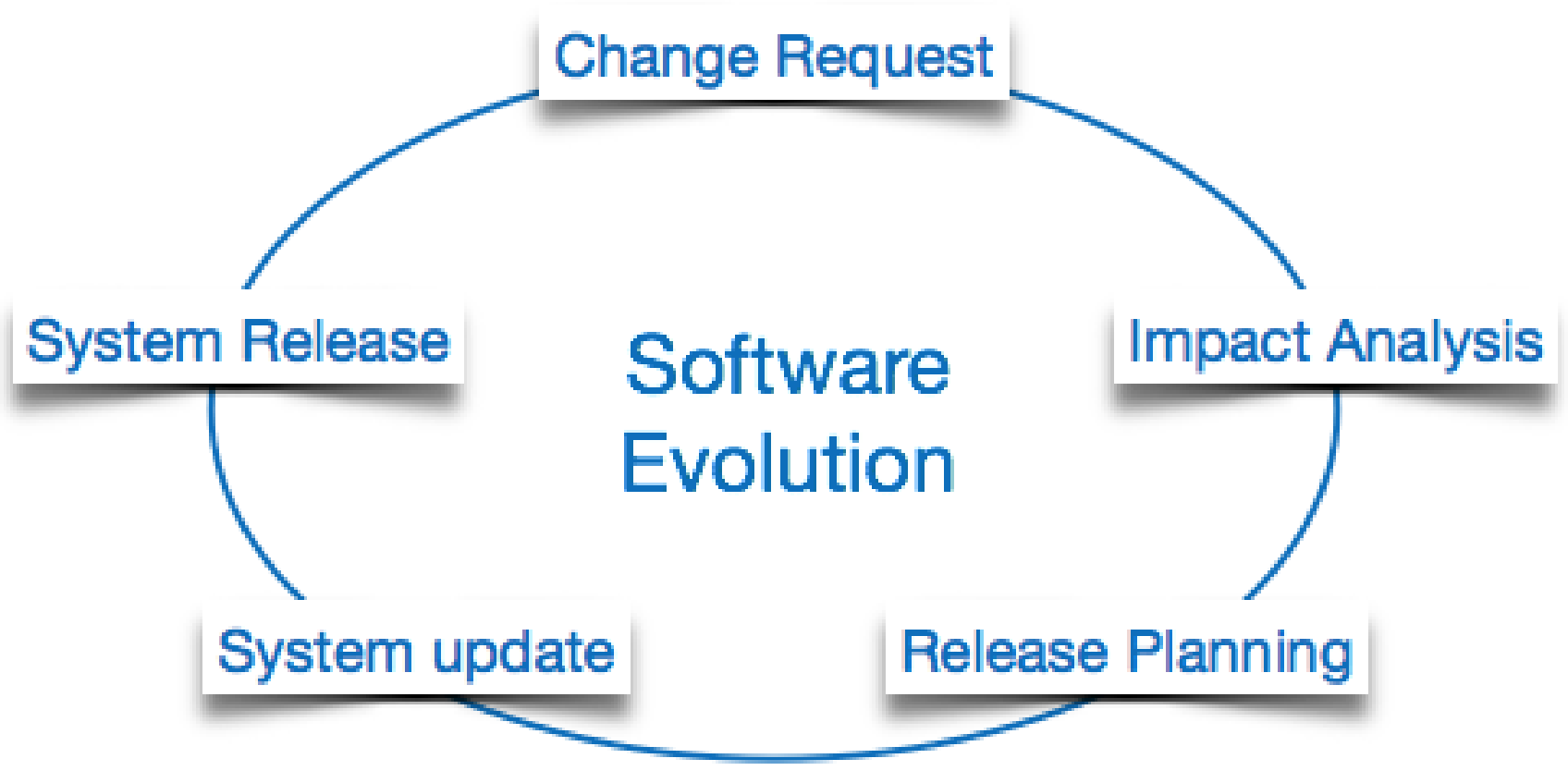
5. Testing

- The testing phase ensures that the software requirements are in place and that the software works as expected.
- When a defect is identified, testers inform the developers.
- If the defect is valid, developers resolve it and create a new version of the software which then repeats the testing phase.
- The cycle continues until all defects are mitigated and the software is ready for deployment into the production environment.

6. Deployment and Maintenance.

- Once the software is error free, it is deployed into the operating environment.
- At this point, customers use the software.
- If there are any issues, they will be brought to the attention of the maintenance team
- They work to resolve them immediately.

Software Evolution



Waterfall Model

Strengths

- Simple and easy to manage– each phase has specific deliverables.
- Milestones are better understood
- Sets requirements stability
- Works well for smaller projects where requirements are very well
- understood.
- Works well when quality is more important than cost or schedule.

Weaknesses

- No working software is produced until end.
- High uncertainty.
- Delays discovery of serious errors.
- After project requirements are gathered, there is no formal way to make changes to the requirements.
- It might not a good model for
- complex projects or projects that take
- more than a few months to complete.

Waterfall model may be applicable to projects where:

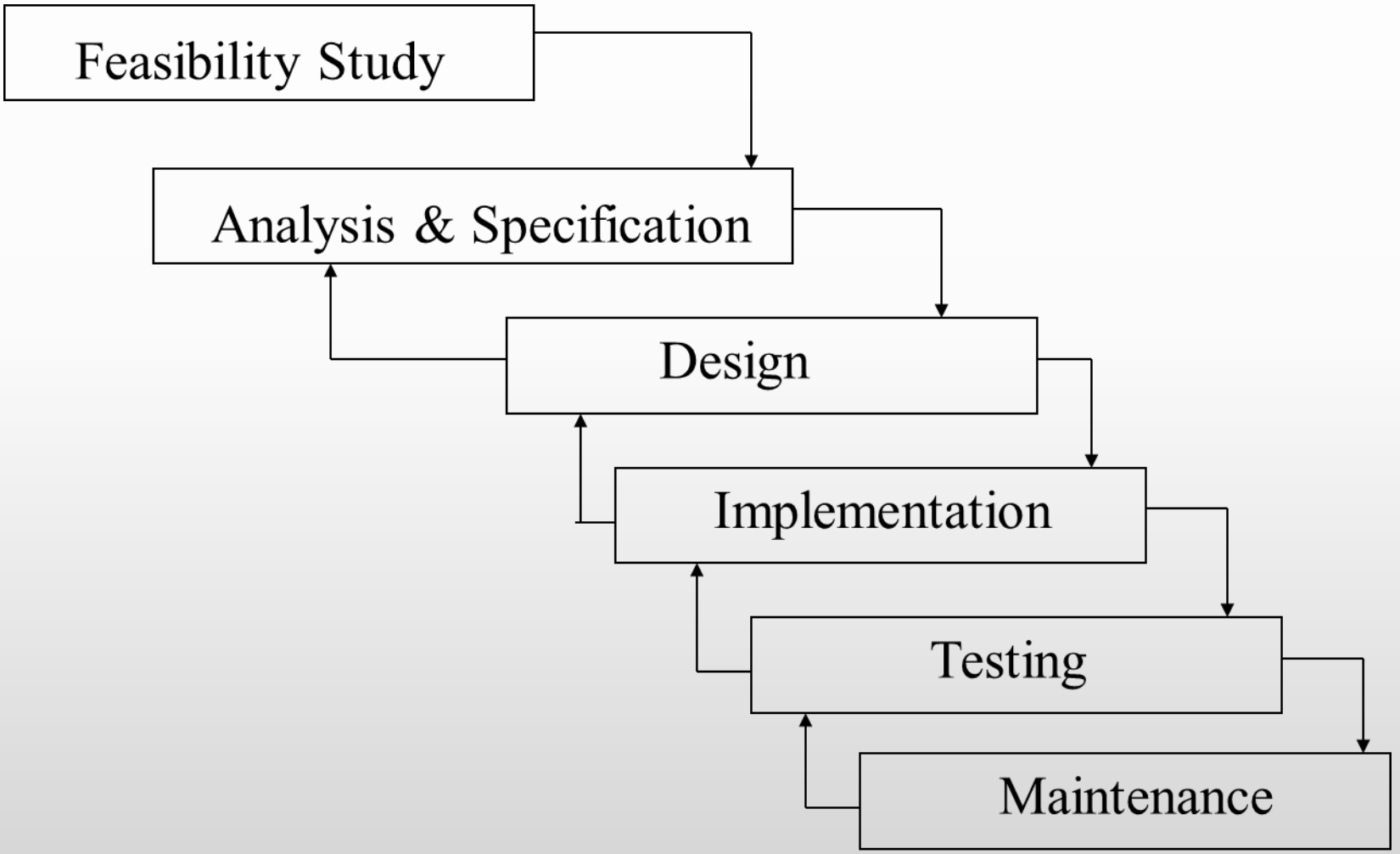
- Software requirements clearly defined and known
- Software development technologies and tools is well known
- New version of the existing software system is created
- Product definition is stable

2
Iterative Model

Iterative Waterfall Model

- The classical waterfall model is idealistic:
  - It assumes that no defects are introduced during any of the development phases.
- In practice, defects are introduced during every phase of the software life cycle:
  - Hence feedback paths must be added to the classical waterfall model.
- The resulting Iterative Waterfall Model is one of the most widely used process models….

Iterative Waterfall Mode



Why Iterative Waterfall Model?

- Defects are often detected late in the software life cycle:
  - For example, a missing requirement might go unnoticed until the system testing phase

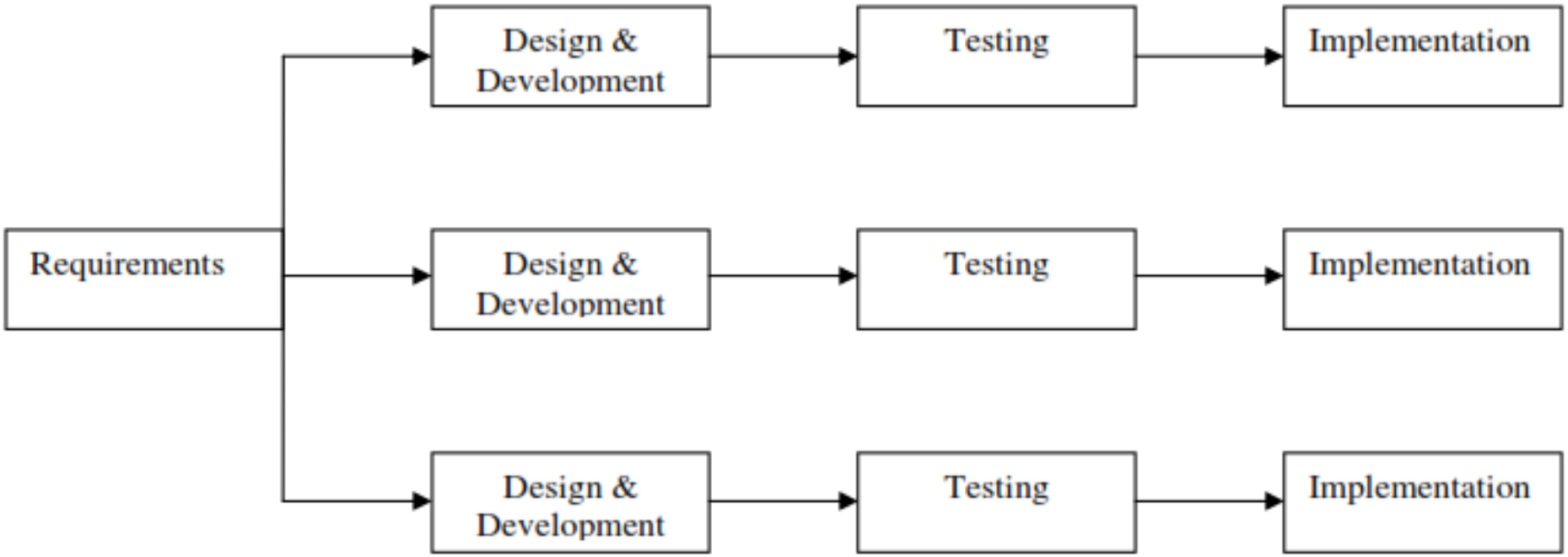
- Once a defect is detected in the Classical Waterfall Model, the team must:
  - Return to the phase in which the defect was introduced
  - Redo some of the work from that phase, and ALL subsequent phases
  - This is time-consuming and expensive!

3
Incremental Model

Incremental model

- Incremental model is an evolution of waterfall model.
- The product is designed, implemented, integrated and tested as a series of incremental builds.
- The incremental model prioritizes requirements of the system and then implements them in groups.
- Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.
- It is the process of constructing a partial implementation of a total system and slowly adding increased functionality or performance.

Incremental model



Incremental model

Strengths

- Generates working software quickly and early during the software life cycle.
- More flexible - less costly to change scope and requirements.
- Easier to test and debug during a smaller iteration.
- Easier to manage risk because risky pieces are identified and handled during its iteration.
- Lowers initial delivery cost
- Risk of changing requirements is reduced

Weaknesses

- Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.
- Requires early definition of a complete and fully functional system to allow for the definition of increments.
- Each phase of an iteration is rigid and do not overlap each other.
- Total cost of the complete system is not lower.
- Requires good planning and design

Incremental model may be applicable to projects where:

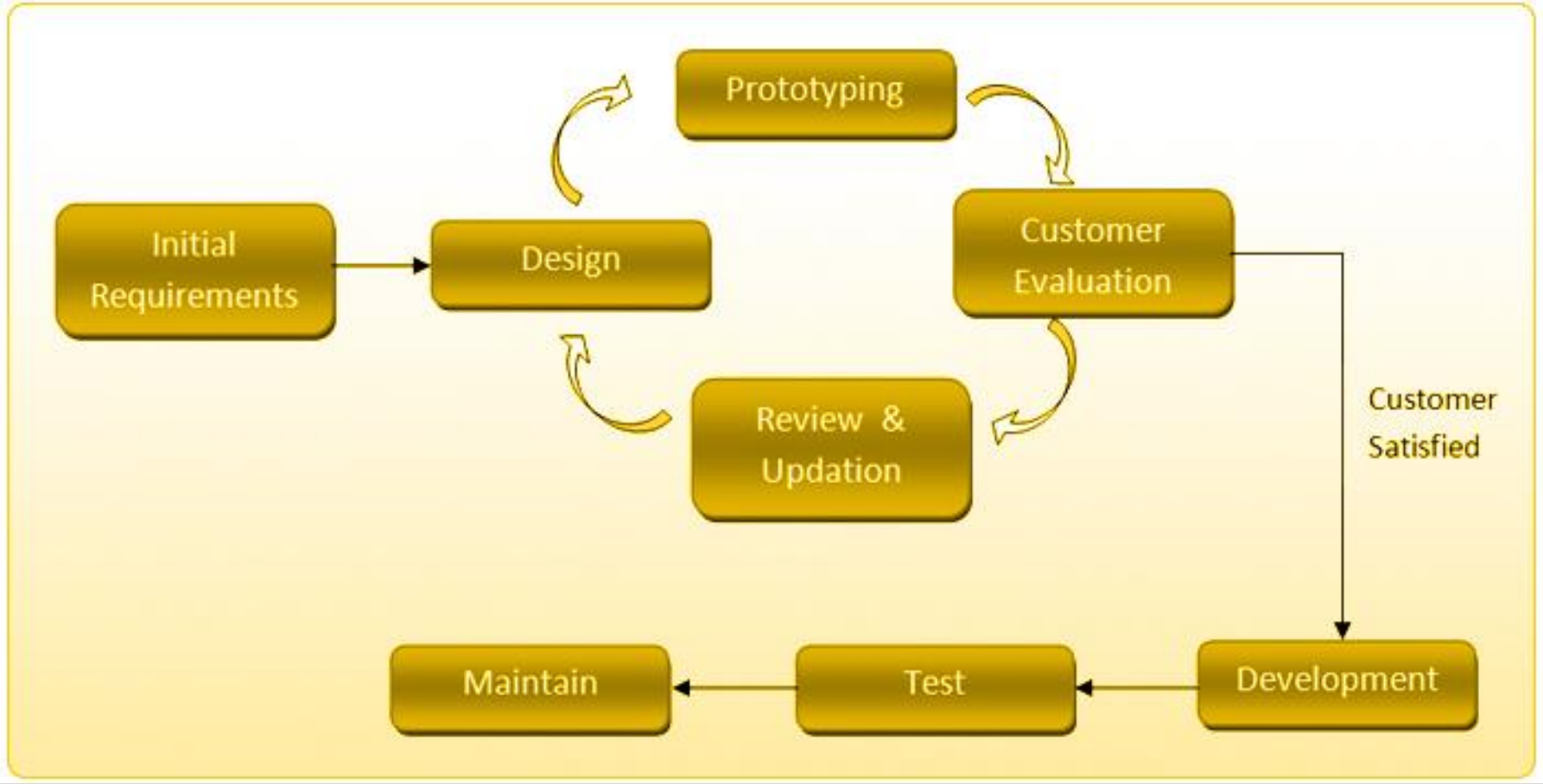
- On projects which have lengthy development schedules
- A need to get basic functionality to the market early
- Most of the requirements are known up-front but are expected to evolve over time
- On a project with new technology
- Software Requirements are well defined, but realization may be delayed.
- The basic software functionality are required early

4
Prototyping Model

Prototype Model

- The prototype model is used to overcome the limitations of waterfall model.
- In this model, instead of freezing the requirements before coding or design, a prototype is built to clearly understand the requirements.
- This prototype is built based on the current requirements.
- Through examining this prototype, the client gets a better understanding of the features of the final product.

Prototype Model



Prototyping Model

Strengths

- Ability clarify user's expectations for the system to be developed
- Prototype stimulates awareness of additional needed functionality
- Better user satisfaction
- Quicker user feedback is available leading to better solutions.

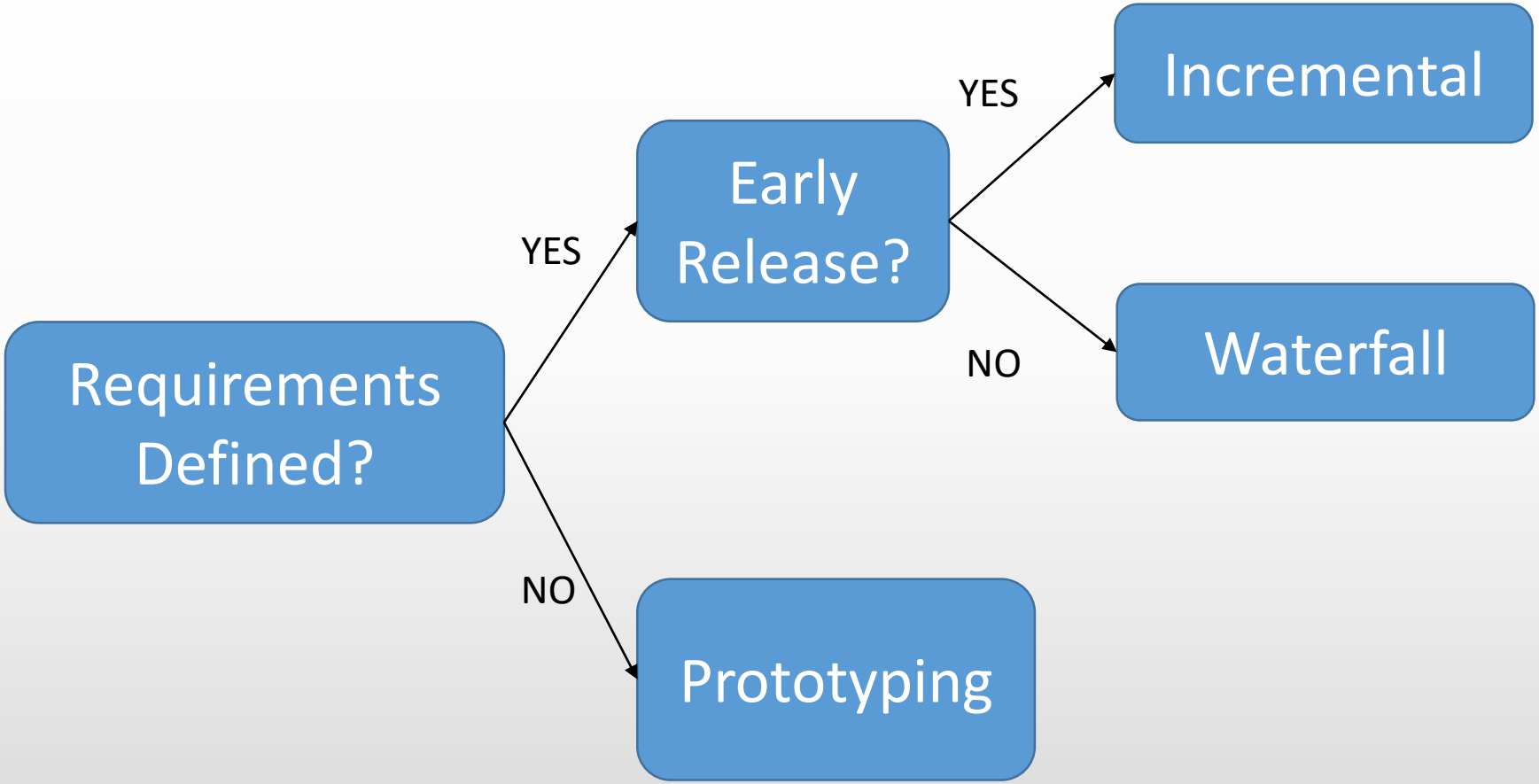
Weaknesses

- The system as scope may expand beyond original plans.
- Overall maintainability may be overlooked.
- The customer may want the prototype delivered.
- Process may continue forever (scope creep)

Prototyping may be applicable to projects where:

- Requirements are unstable or have to be clarified
- Develop user interfaces
- Short-lived demonstrations
- New, original development

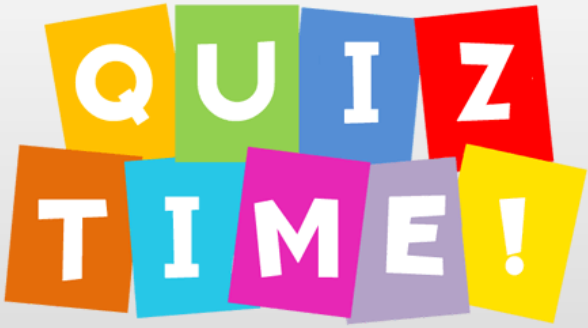
Selection of Approach



Question

- A company is developing an advanced version of their current software available in the market, what model approach would they prefer ?

a) Waterfall
b) Incremental
c) Prototyping

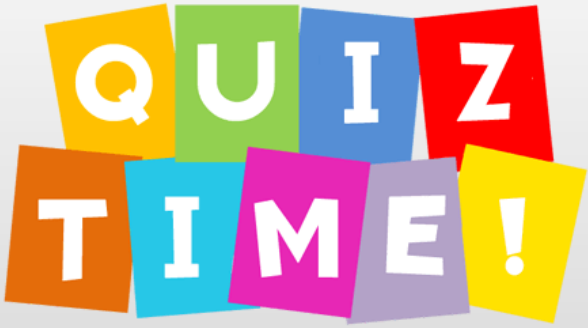


SLIIT
Discover Your Future

Question

Which of the following life cycle model can be chosen if the development team has less experience on similar projects?

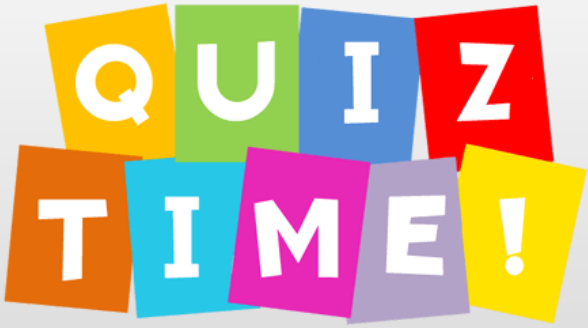
a) Spiral
b) Waterfall
d) Iterative



Question

- Which of the following models will not be able to give the desired outcome if user's participation is not involved?

a) Waterfall
b) Spiral
c) Prototyping



5
Spiral Model

Spiral Model

- The spiral model combines the idea of iterative development (prototyping) with the systematic, controlled aspects of the waterfall model.
- It allows for incremental releases of the product, or incremental refinement through each time around the spiral.
- The spiral lifecycle model allows for elements of the product to be added in when they become available or known.
- This assures that there is no conflict with previous requirements and design.

Spiral Model

- Documents are produced when they are required, and the content reflects the information necessary at that point in the process. Like the product they define, the documents are works in progress. The idea is to have a continuous stream of products produced and available for user review.

- The development spiral consists of four quadrants: Quadrant 1: Determine objectives, alternatives, and constraints.
- Quadrant 2: Evaluate alternatives, identify, resolve risks.
- Quadrant 3: Develop, verify, next-level product.
- Quadrant 4: Plan next phases.

Spiral Model

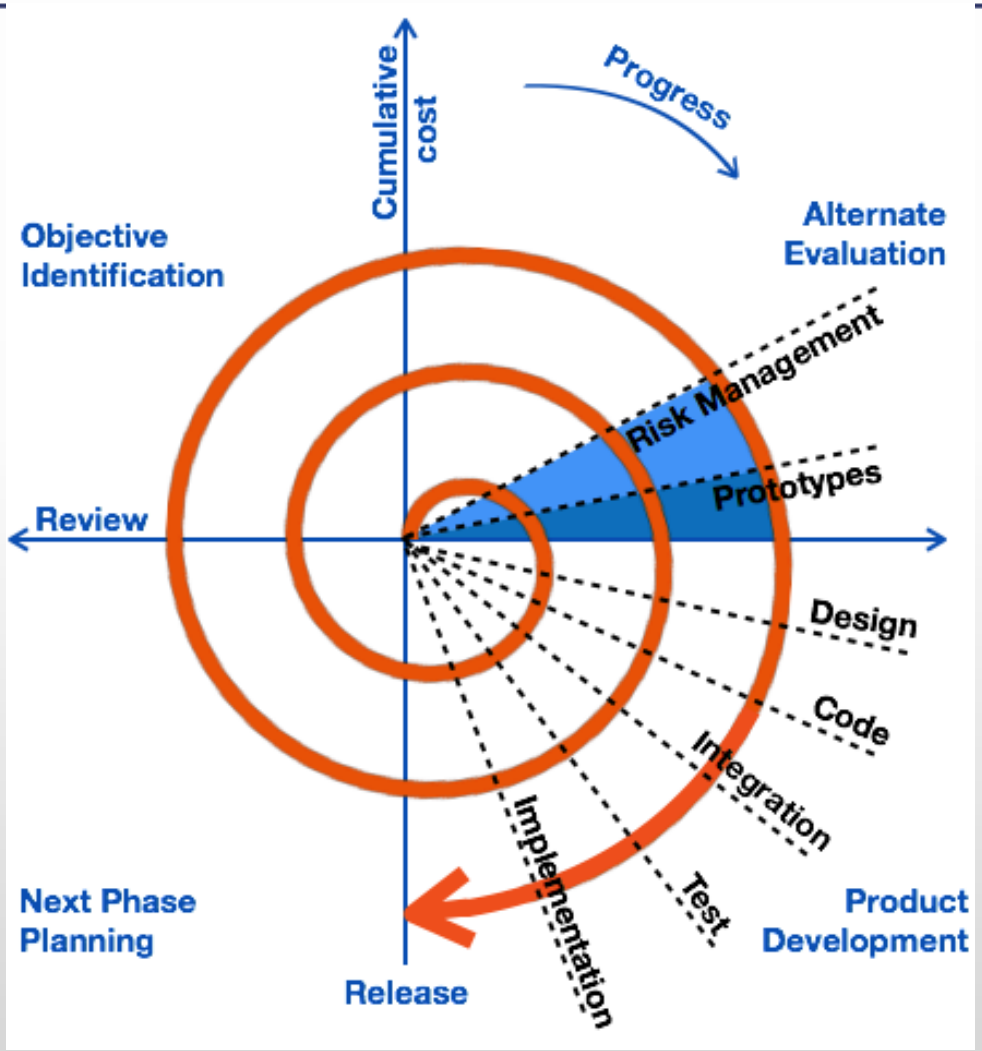
Strengths

- High amount of risk analysis.
- Good for large and mission critical projects
- Software is produced early in the software life cycle.
- Provides early indication of insurmountable risks, without much cost.
- Users see the system early because of rapid prototyping tools.
- Critical high-risk functions are developed first
- The design does not have to be perfect
- Early and frequent feedback from users
- Cumulative costs assessed frequently

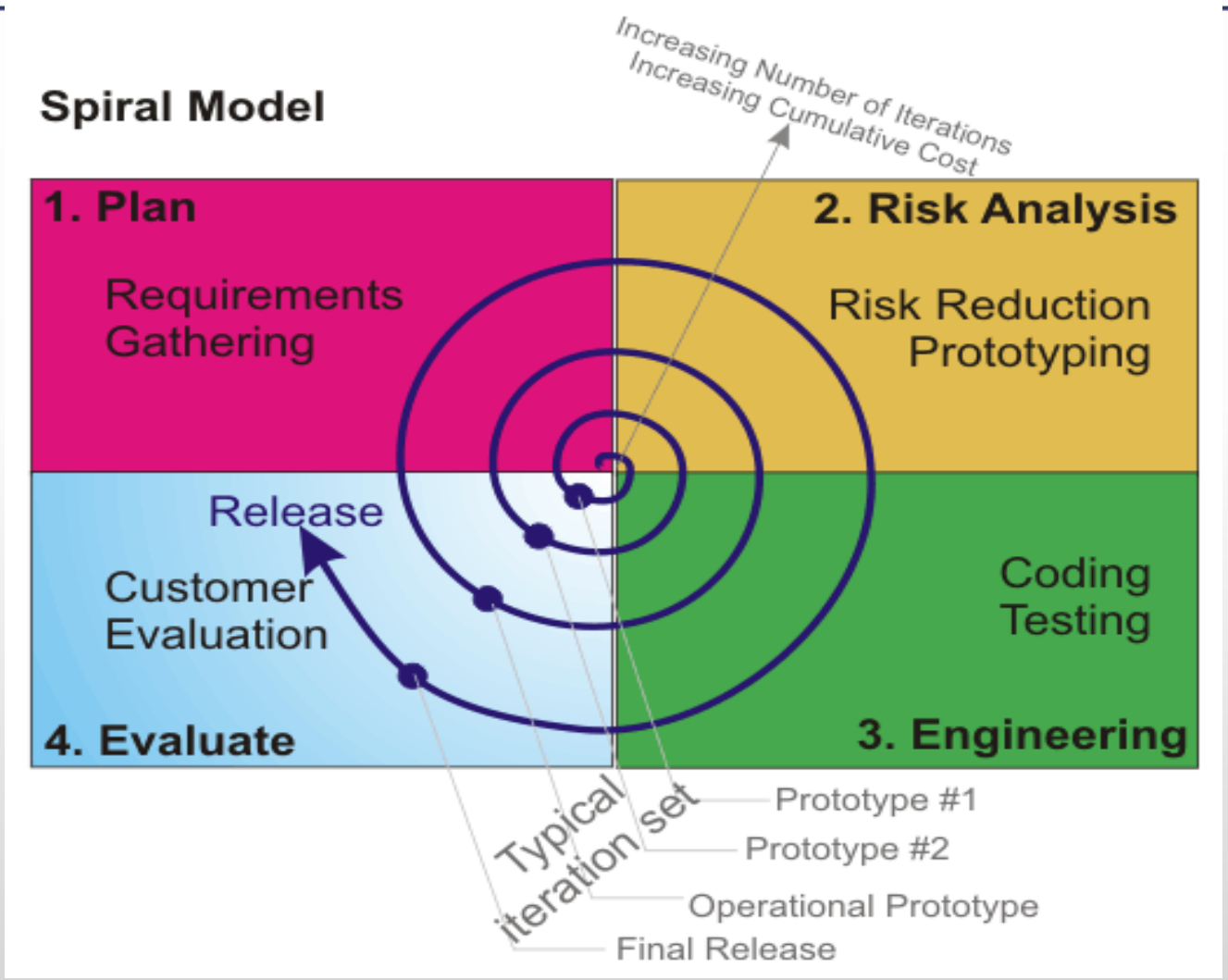
Weaknesses

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- success is highly dependent on the risk analysis phase.
- Doesn’t work well for smaller projects.

Spiral Model



Spiral Model



Spiral model may be applicable to projects where:

- When creation of a prototype is appropriate
- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- Significant changes are expected (research and exploration)

6
Unified Process

Unified Process

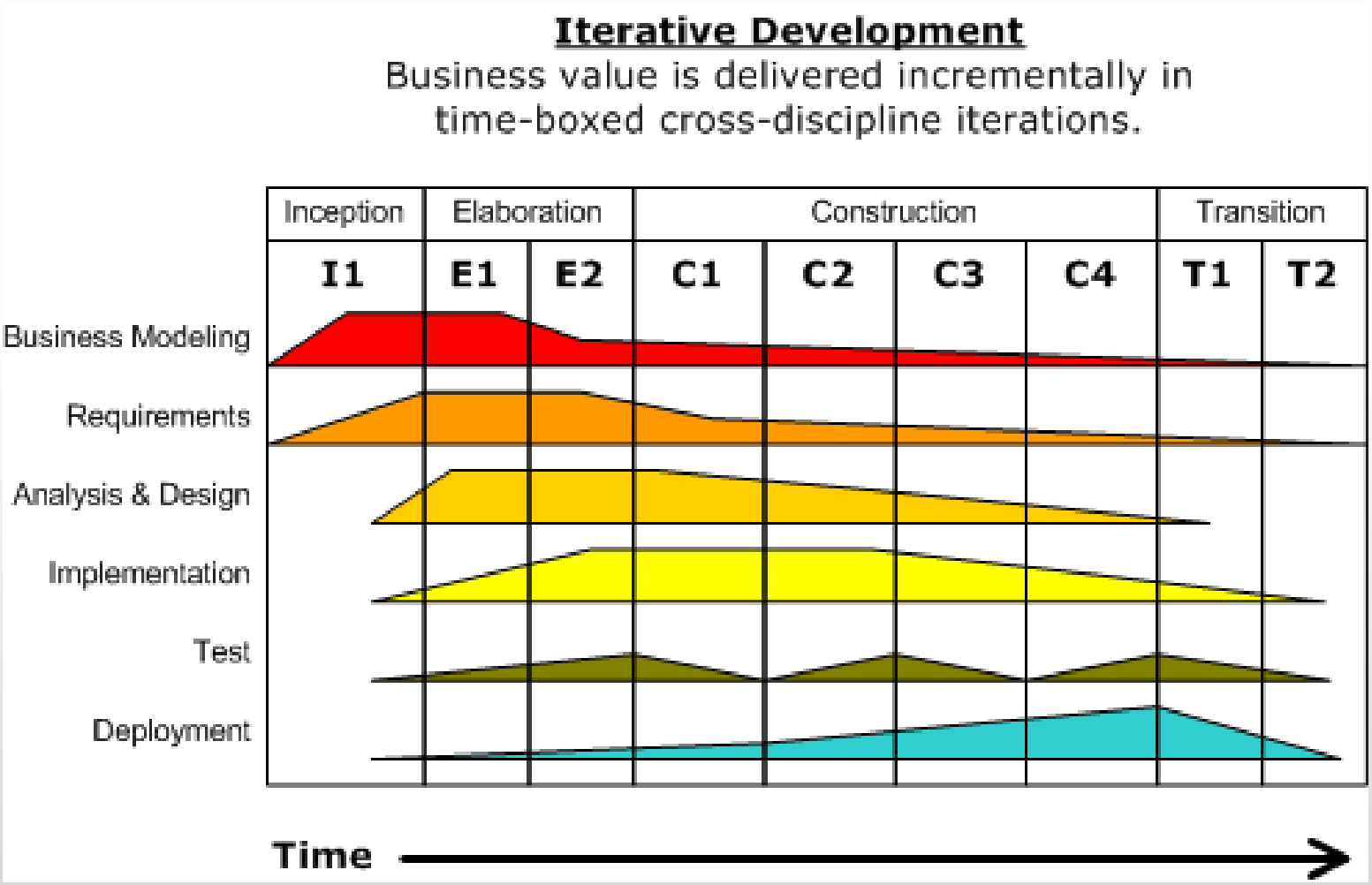
- The **Unified Process** is a popular iterative and incremental software development process framework.
- Examples
  - Rational Unified Process (RUP).
  - OpenUP
  - Agile Unified Process.

Unified Process

- The Unified Process (UP) is a use-case-driven, architecture-centric, iterative and incremental development process framework that leverages the Object Management Group's (OMG) UML.

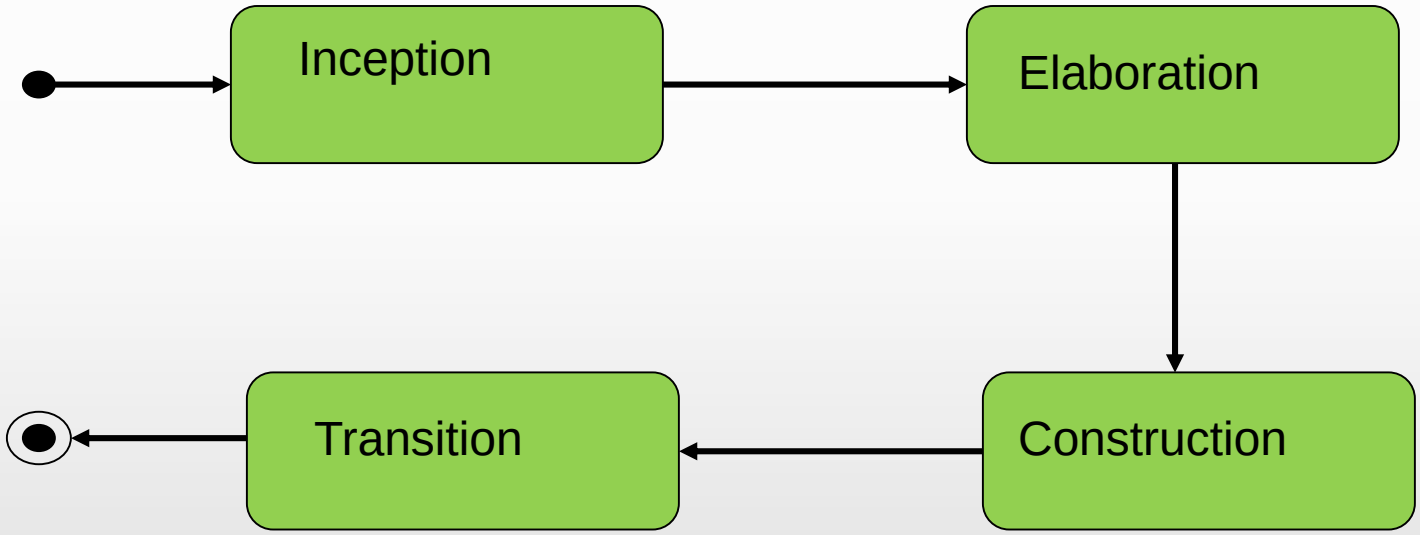
- The UP is broadly applicable to different types of software systems, including small and large-scale projects having various degrees of managerial and technical complexity, across different domains.

Unified Process



Unified Process

The Unified Process divides the project into four phases:



UP Lifecycle – single phase workflow
(drawn as a UML Statechart!)

Unified Process

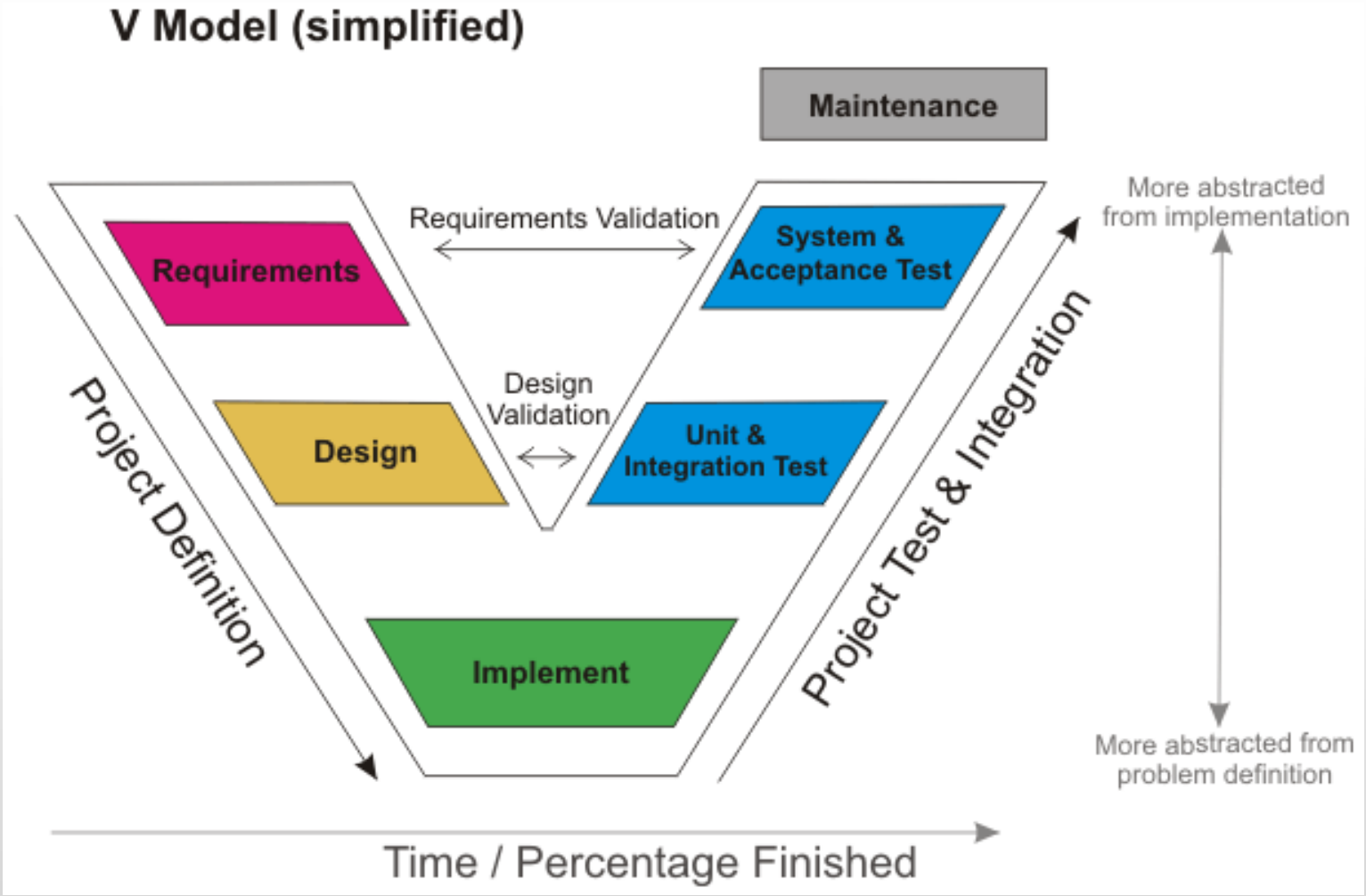
- **Inception** – identify core use cases, and use to make architecture and design tradeoffs. Estimate and schedule project from derived knowledge.
- **Elaboration** – capture detailed user requirements. Make detailed design, decide on build vs. buy.
- **Construction** – components are bought or built, and integrated.
- **Transition** – release a mature version that satisfies acceptance criteria.

7
V Model

V Model

- **V-model** may be considered an extension of the waterfall model.
- The V-Model demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.
- The horizontal axes represents time or project completeness (left-to-right).
- The vertical axes represents level of abstraction.

V Model


V Model (simplified)
Maintenance
Requirements
Requirements Validation
System & Acceptance Test
Design
Design Validation
Unit & Integration Test
Implement
Project Definition
Project Test & Integration
More abstracted from implementation
More abstracted from problem definition
Time / Percentage Finished

Question

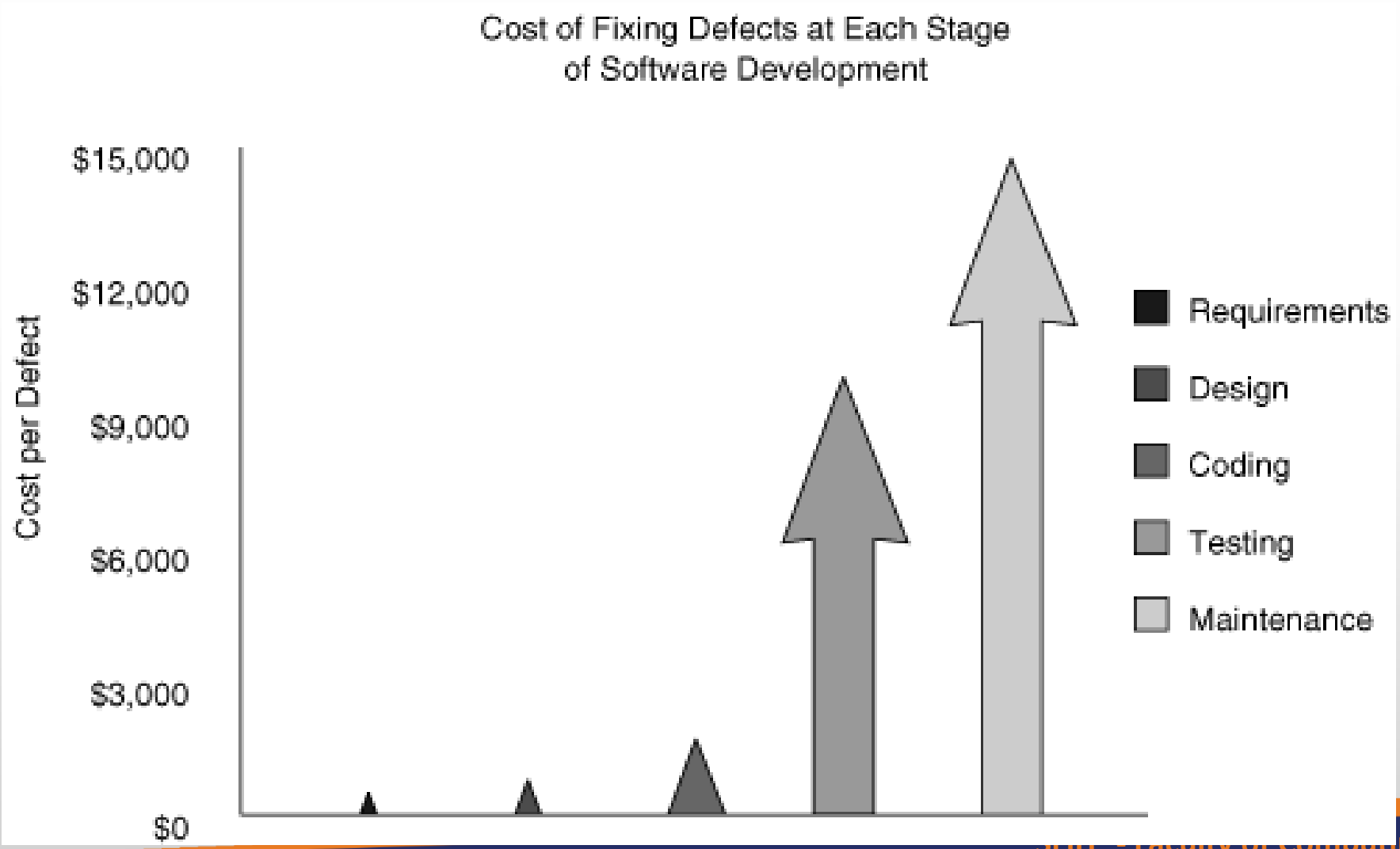
Selection of a model is based on…?

a) Requirements
b) Development team
c) Users
d) Project type and associated risk
e) All of the mentioned

	Waterfall	V-Shaped	Evolutionary Prototyping	Spiral	Iterative and Incremental	Agile
Unclear User Requirement	Poor	Poor	Good	Excellent	Good	Excellent
Unfamiliar Technology	Poor	Poor	Excellent	Excellent	Good	Poor
Complex System	Good	Good	Excellent	Excellent	Good	Poor
Reliable system	Good	Good	Poor	Excellent	Good	Good
Short Time Schedule	Poor	Poor	Good	Poor	Excellent	Excellent
Strong Project Management	Excellent	Excellent	Excellent	Excellent	Excellent	Excellent
Cost limitation	Poor	Poor	Poor	Poor	Excellent	Excellent
Skills limitation	Good	Good	Poor	Poor	Good	Excellent
Documentations	Excellent	Excellent	Good	Good	Excellent	Poor
Component reusability	Excellent	Excellent	Poor	Poor	Excellent	Poor

8
Secure Software Development

Cost of fixing defect at each stage



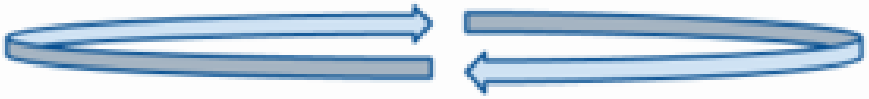
Software security

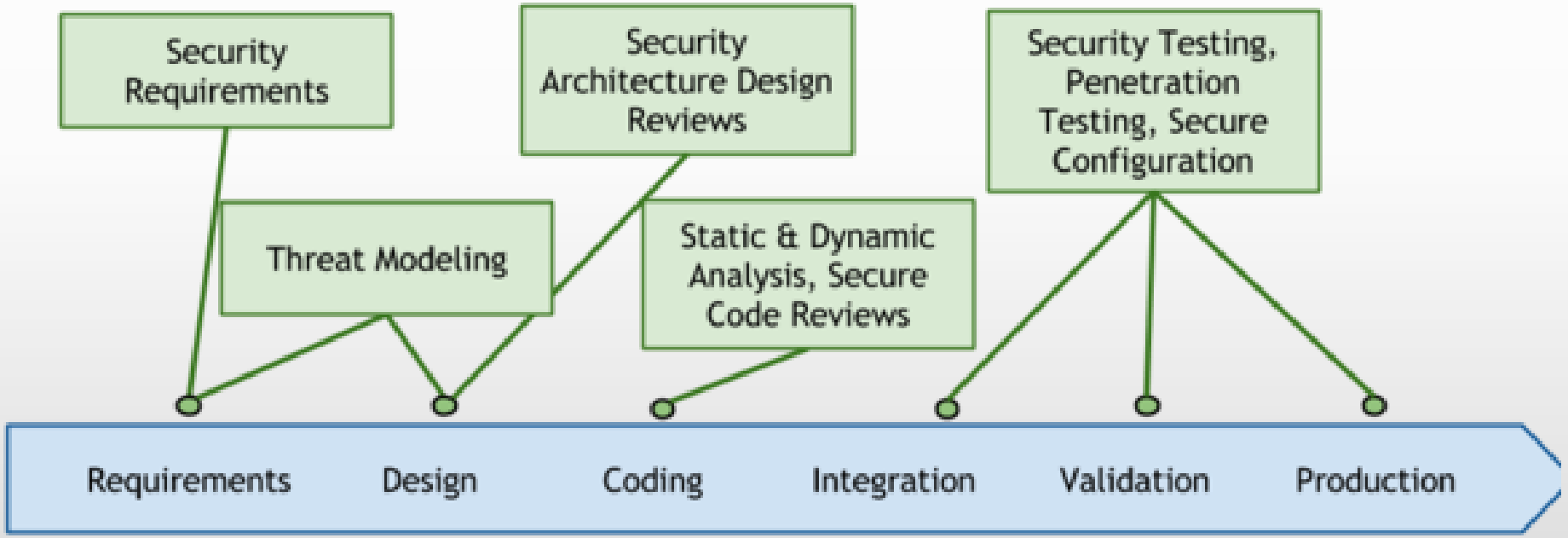
- It is about
  - Understanding software security risks and how to manage them
  - Thinking security early in the software lifecyle
  - Knowing and understanding common problems
  - Designing for security
  - Subjecting all software artifacts to thorough risk analyses and testing

Secure Development Lifecycle

- Firms have recognized the need for secure code.
- Adding security later tends to cost exponentially more than implementing it from the start.
- Security should be an issue that is addressed throughout the development process.
- The SDL accounts for security in each of its four major phases:
  - Requirements phase
  - Design phase
  - Coding phase
  - Testing phase

Security in the SDLD Process





Software Security Touchpoints

- SSDLC is set up by implementing security assurance activities within an existing development process.

- Example Touchpoints
  - **Security requirements** - Describes functional behavior that enforces security. It can be directly tested and observed. E.g. access control, data integrity, authentication, and wrong password lockouts fall under functional requirements.
  - **Code review (peer review)** -A systematic examination of **code**. It is intended to find mistakes overlooked in the initial development phase.
  - **Penetration testing (pen testing) -**Practice of **testing** a computer system, network or Web application to find vulnerabilities that an attacker could exploit.

Software Security Touchpoints

- **Abuse cases** - An adaptation of use case. *A complete abuse case defines an interaction between an actor and the system that results in harm to a resource associated with one of the actors, stakeholders, or the system itself.*
- **Threat modeling** - A structured approach that enables to identify, quantify, and address the security risks associated with an application.
- **Risk-based testing** (RBT) - A type of softwaretesting that functions as an organizational principle used to prioritize the tests of features and functions in software, based on the risk of failure, the function of their importance and likelihood or impact of failure.
- **Architectural risk analysis -** Architectural risk assessment is a risk management process that identifies flaws in a software architectureand determines risks to business information assets that result from those flaws.
- **Security operations**

Advantages of SSDLC

- More secure software due to the fact that security is a continuous concern.
- Stakeholder awareness of security considerations.
- Early flaw detection in the system.
- Cost reduction as a result of early detection and resolution of issues.
- Overall reduction of intrinsic business risks for the organization.

Activities

Which life cycle is suitable?

Mini-Case 1

- The project is to develop an inventory control system for a new super market in town. It should have all the regular functionalities such as adding new stocks, getting reports such as inventory re-order report and daily sales report etc. A project team has previous experience developing inventory systems for other clients.

Mini-Case 2

- A library management system is required for a National Library with a number of branches in many cities. System should link to a number of other online libraries, databases, journals and university libraries through web and manage different user subscriptions. The project team has little experience in developing library systems before. However there is not much pressure on time.

Mini-Case 3

- A Project is to develop a complete system for a new bank. System will have many users, interrelationships, and functions. The project has few risks related to requirements definition. These risks needs to be reduced.

References

- https://www.tutorialspoint.com/software_engineering/software_development_life_cycle.htm
- *Software Security - Building Security In,* Gary McGraw, Addition-Wesley Software Security Series, ISBN: 0-321-35670-5
- *Building Secure Software: How to avoid the Security Problems the Right Way,* John Viega, Gary McGraw, Addison-Wesley, 2002