Software Process

What is well engineered software?

If the software system does what the user wants, and can be made to continue to do what the user wants, it is well engineered.

- Any well engineered software system should have the following attributes:
- Be easy to maintain
- Be reliable
- Be efficient
- Provides an appropriate user interface

The development of software must make trade-offs between these attributes.

Distribution of software effort

The typical life-span for a typical software product is 1 to 3 years in development and 5 to 15 years in use. The distribution of effort between development and maintenance has been variously reported depending on the type of software as 40/60, 30/70 and 10/90.

Maintenance:

- **Corrective**: Even with the best quality of software, it is likely that customer will uncover defect in software. Corrective maintenance changes the software to correct the defects.

- **Adaptive:** Over time, the original environment (CPU, OS, business rules, external product character etc.) for which the software was developed may change. Adaptive maintenance results in modification to the software to accommodate the change to its environment.

- **Perfective**: As the software is used, the customer / user will recognize additional function that will provide benefit. Perfective maintenance extends the software beyond its original functional requirements.

Software Process

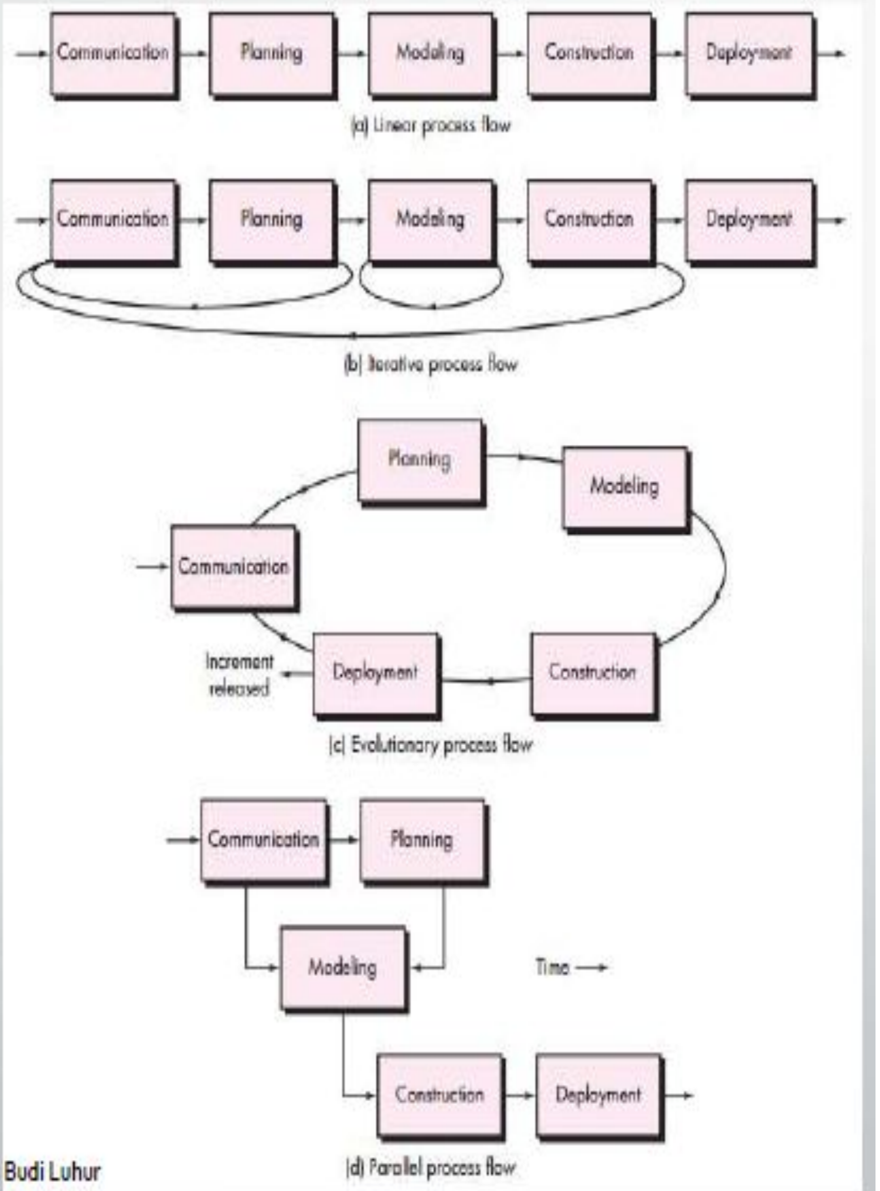
- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective

[Sommerville, 2011]

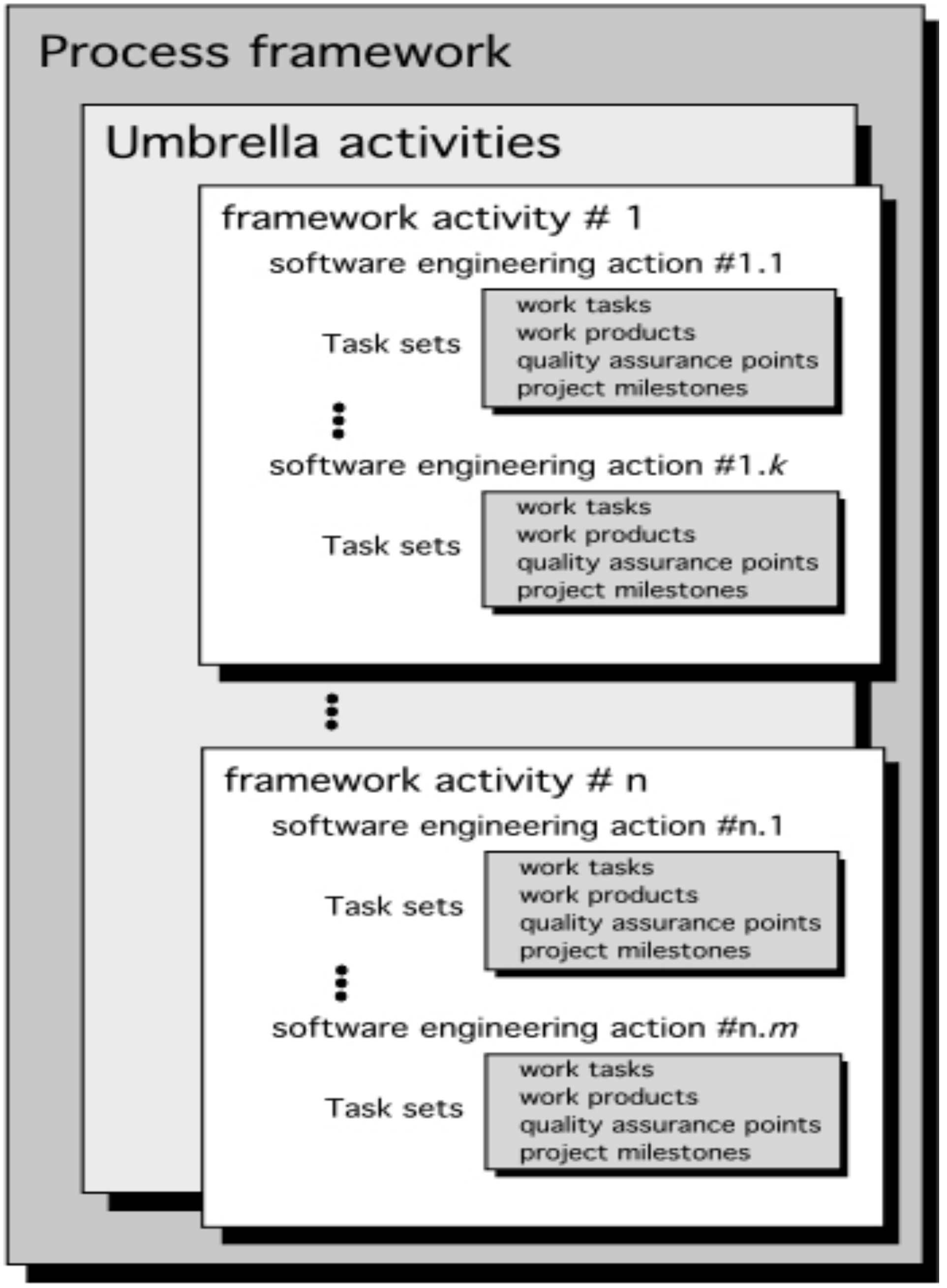
Types of Process Models

- Universal
  - Describe the basic process steps and provide general guidance on their role and order (e.g., Waterfall and Spiral Model). Permit global understanding and provide a framework for policies.
- Worldly
  - Guide the sequence of tasks, define task prerequisites and results, specify who does what when, models anticipate results, measure and key checkpoints. Guide daily work.
- Atomic
  - Precise data definitions, algorithmic specifications, information flows and user procedures. Atomic process definitions are often embodied in process standards and conventions. Provide atomic detail for training and task mechanization.

Process Flow



Software process



Task Set

A task set defines the actual work to be done to accomplish the objectives of a software engineering action.

For a small, relatively simple project, the task set for requirements gathering might look like this:

1. Make a list of stakeholders for the project.
2. Invite all stakeholders to an informal meeting.
3. Ask each stakeholder to make a list of features and functions required.
4. Discuss requirements and build a final list.
5. Prioritize requirements.
6. Note areas of uncertainty

Task Set

For a larger, more complex software project:

1. Make a list of stakeholders for the project.
2. Interview each stakeholder separately to determine overall wants and needs.
3. Build a preliminary list of functions and features based on stakeholder input.
4. Schedule a series of facilitated application specification meetings.
5. Conduct meetings.
6. Produce informal user scenarios as part of each meeting.
7. Refine user scenarios based on stakeholder feedback.
8. Build a revised list of stakeholder requirements.
9. Use quality function deployment techniques to prioritize requirements.
10. Package requirements so that they can be delivered incrementally.
11. Note constraints and restrictions that will be placed on the system.
12. Discuss methods for validating the system.

Software Development Life cycle (SDLC)

A life cycle model prescribes the different activities that need to be carried out to develop a software product and sequencing of these activities.

Also referred to as Systems Development Life cycle.

Every software product starts with a request for the product by the customer.- Production conception.

The software life cycle can be considered as the business process for software development and therefore is often referred to as a Software process. (SLCM).

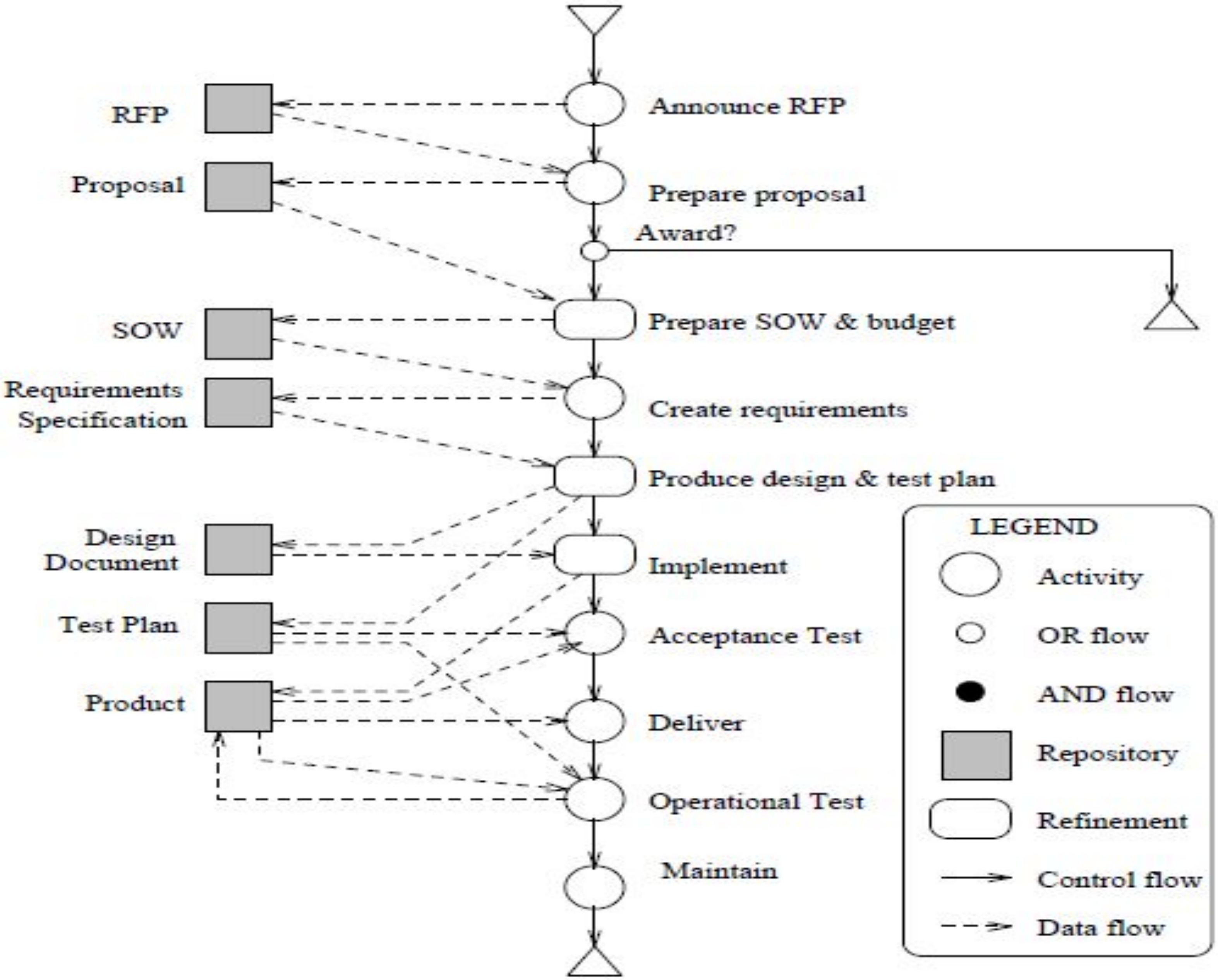
Process models – More detailed and precise life cycle activities.

- Why use a life cycle model?

- Encourages development of software in a systematic and disciplined manner
- S.D organisations have realized that adherence to a suitable well-defined life cycle model helps to produce good quality products and that too without time and cost overruns.

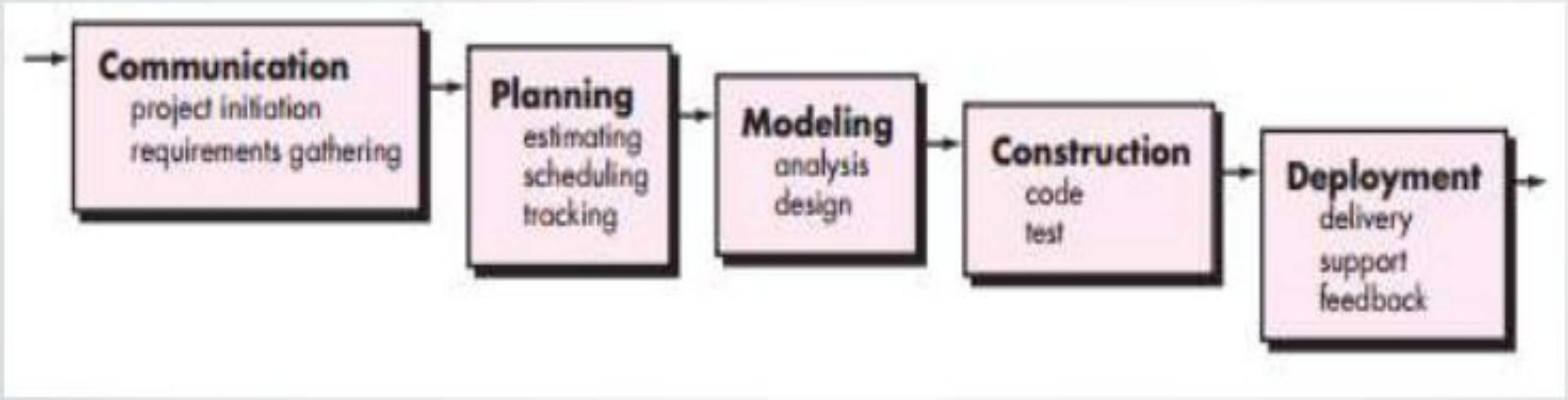
- Why document a life cycle model?

- A documented life cycle model, besides preventing misinterpretations that occur when the life cycle model is no adequately documented, also helps to identify inconsistencies, redundancies, and omissions in the development process

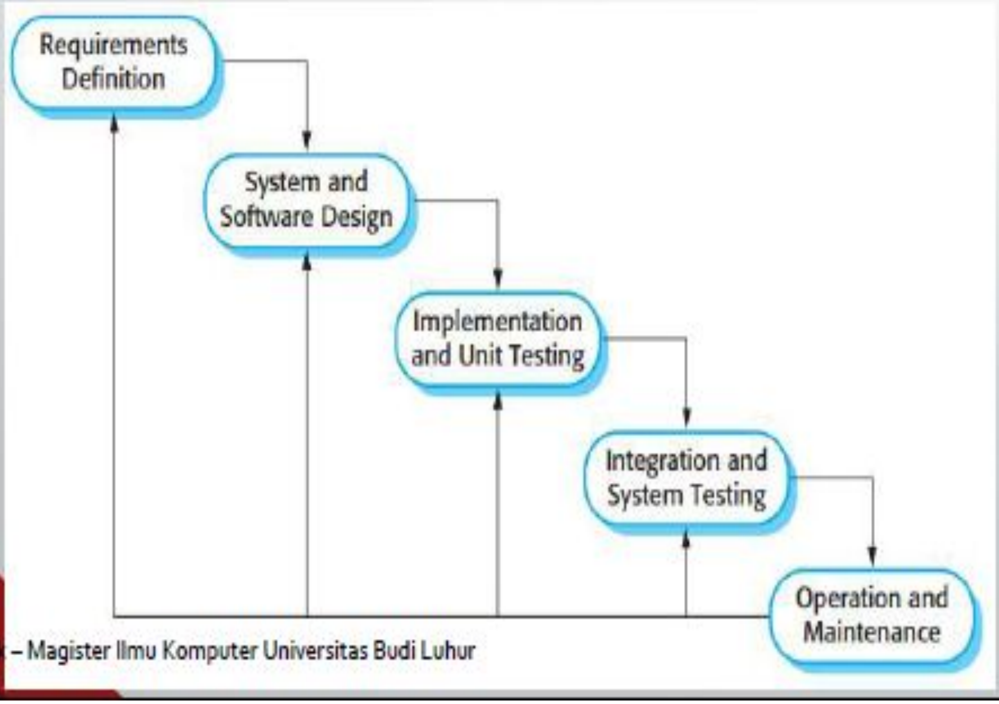

RFP
Proposal
SOW
Requirements Specification
Design Document
Test Plan
Product
Announce RFP
Prepare proposal
Award?
Prepare SOW & budget
Create requirements
Produce design & test plan
Implement
Acceptance Test
Deliver
Operational Test
Maintain
LEGEND
Activity
OR flow
AND flow
Repository
Refinement
Control flow
Data flow

- **Different stages in a life cycle model:** After **Product conception.** The stages are: (**Life cycle phase**)
  - **Feasibility study stage**
  - **Requirements analysis and specification.**
  - **Design**
  - **Coding**
  - **Testing and**
  - **Maintenance.**
- A SDLC is a series of identifiable stages that a software product undergoes during its lifetime.
- A SDLC is a descriptive and diagrammatic representation of the software life cycle.
- A life cycle model maps the different activities performed on a software product from its beginning to retirement into a set of life cycle phases.

The Waterfall Model



[Pressman, 2010]


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

[Sommerville, 2011]

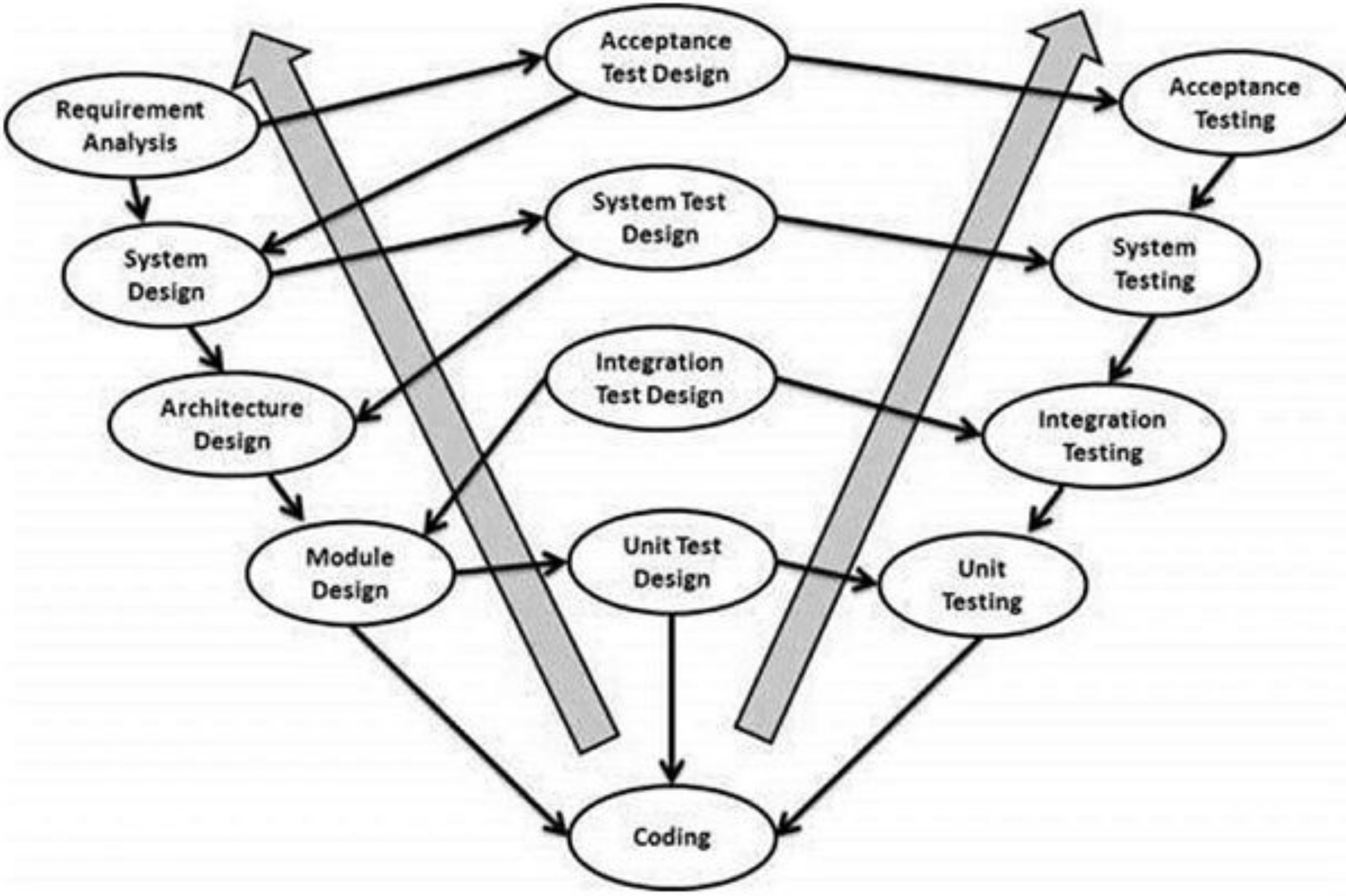
Waterfall Model Problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

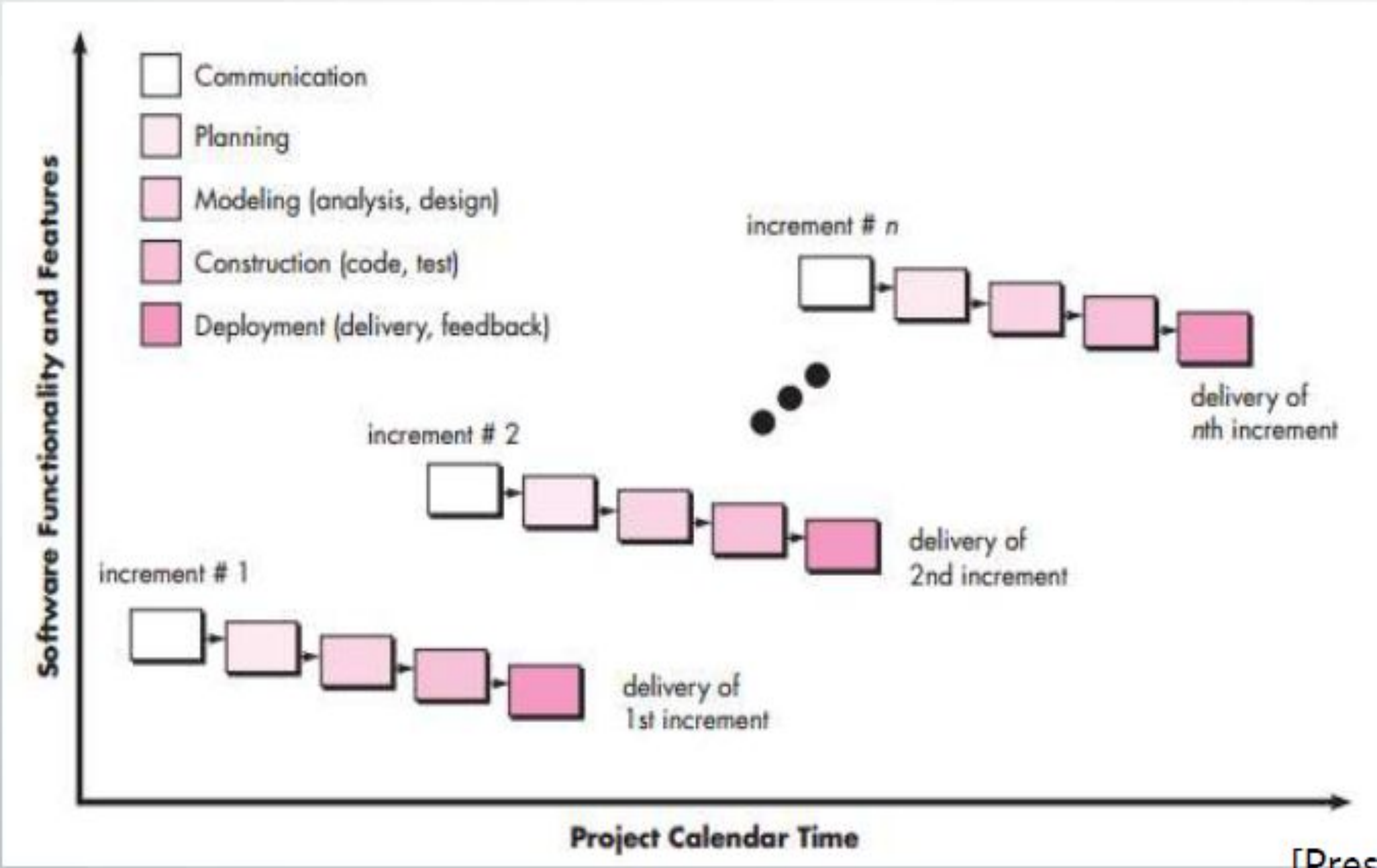
Example of Waterfall model

1. **The NASA Space Program**: The Waterfall model was initially developed for large-scale, complex projects like the NASA Space Program. The Waterfall model was used to develop the software for the Apollo space missions in the 1960s and 1970s.
2. **The Manhattan Project**: The Waterfall model was also used in the development of the atomic bomb during World War II. The project was completed on time and within budget, despite its complexity and secrecy.
3. **The Automated Teller Machine (ATM):** The Waterfall model was used to develop the software for the first ATM machines in the 1970s. This project was also completed on time and within budget, and the resulting technology revolutionized the banking industry.
4. **The Hubble Space Telescope**: The software for the Hubble Space Telescope was developed using the Waterfall model. Despite some initial technical difficulties, the project was eventually successful and the telescope has made many groundbreaking discoveries.
5. **The Microsoft Windows Operating System**: Microsoft used the Waterfall model to develop several versions of its Windows operating system. While there have been some well-publicized bugs and glitches in the software over the years, Windows is still one of the most widely used operating systems in the world.

The following illustration depicts the different phases in a V-Model of the SDLC.



The Incremental Model


Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # 1
delivery of 1st increment
increment # 2
delivery of 2nd increment
increment # n
delivery of nth increment
Software Functionality and Features
Project Calendar Time

[Pres

The Incremental Model Benefits

- The cost of accommodating changing customer requirements is reduced. The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done. Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible. Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

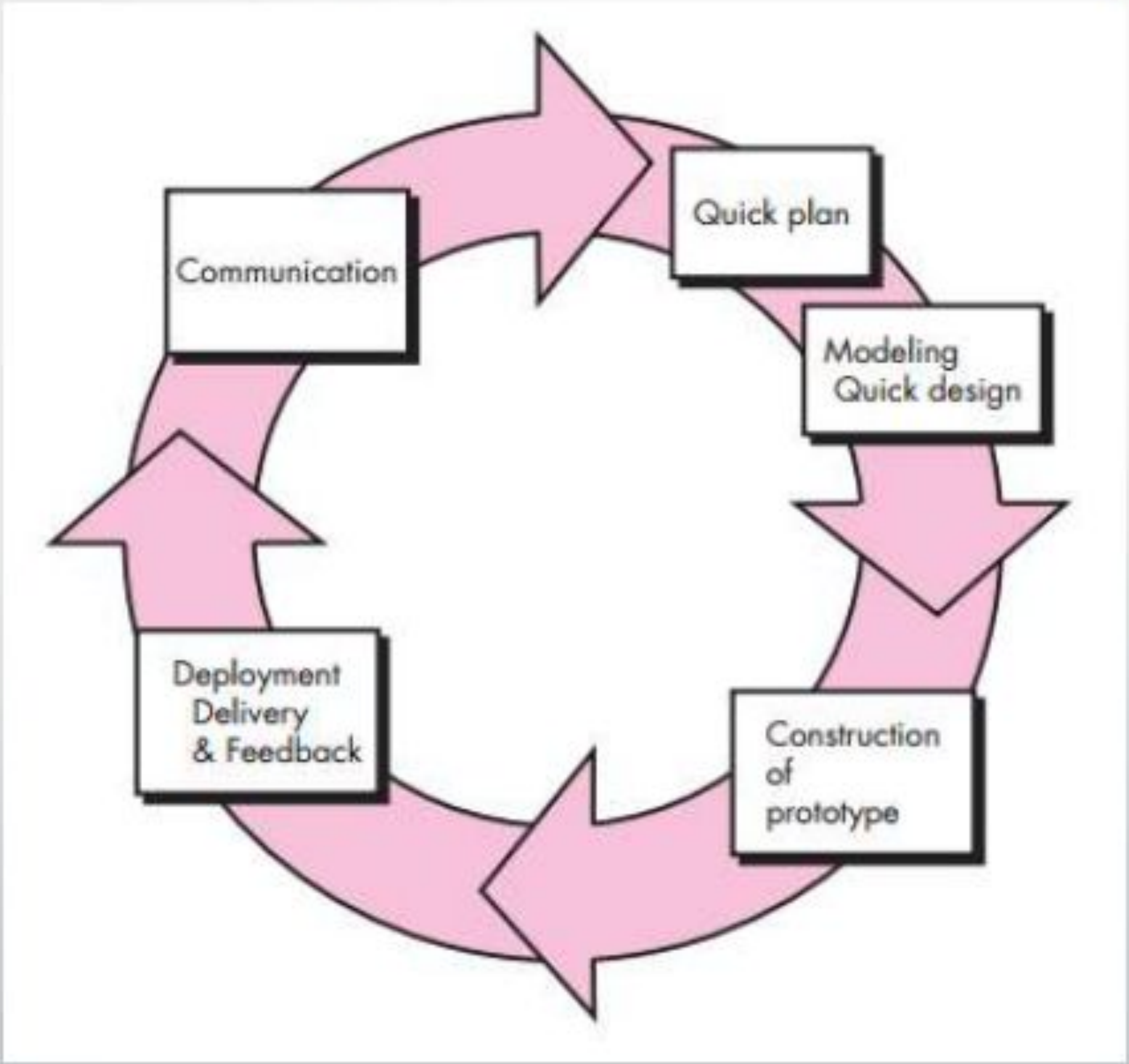
The Incremental Model Problems

- The process is not visible.
    - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- System structure tends to degrade as new increments are added.
    - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.
- The problems of incremental development become particularly acute for large, complex, long-lifetime systems, where different teams develop different parts of the system.

Examples

- 1. **Operating System Development:** Creating a new operating system or a major version update often involves incremental development. Each increment may add new features, improve performance, or enhance security.
- 2. **Game Development:** Building a complex video game is typically done incrementally. Developers may start with basic gameplay mechanics and gradually add levels, characters, and features in successive increments.
- 3. **Web Application Development:** Developing a large web application can benefit from the incremental model. Each increment can focus on specific modules or features like user authentication, payment processing, or content management.
- 4. **Enterprise Software:** Developing comprehensive enterprise software, such as Customer Relationship Management (CRM) or Enterprise Resource Planning (ERP) systems, can be broken down into increments to deliver functionality over time.
- 5. **Content Management Systems (CMS):** Building a CMS may involve incremental development to introduce content editing features, user roles, and plugins incrementally.
- 6. **E-commerce Platforms:** Developing an e-commerce platform may follow an incremental approach, starting with core shopping cart functionality and then adding features like product reviews, inventory management, and payment gateways.
- 7. **Mobile App Development:** Building mobile apps for platforms like iOS and Android often involves releasing versions with essential features first and then gradually adding more features in subsequent updates.
- 8. **Software Infrastructure:** When creating software infrastructure components like databases, messaging systems, or cloud services, developers may use an incremental model to release and improve these services incrementally.
- 9. **Embedded Systems:** Developing software for embedded systems used in devices like IoT (Internet of Things) devices may involve incremental updates to enhance functionality and address issues.
- 10. **Software Prototyping:** In the early stages of software prototyping, an incremental approach allows developers to build and refine prototypes progressively based on user feedback.

Evolutionary Model: Prototyping



[Pressr

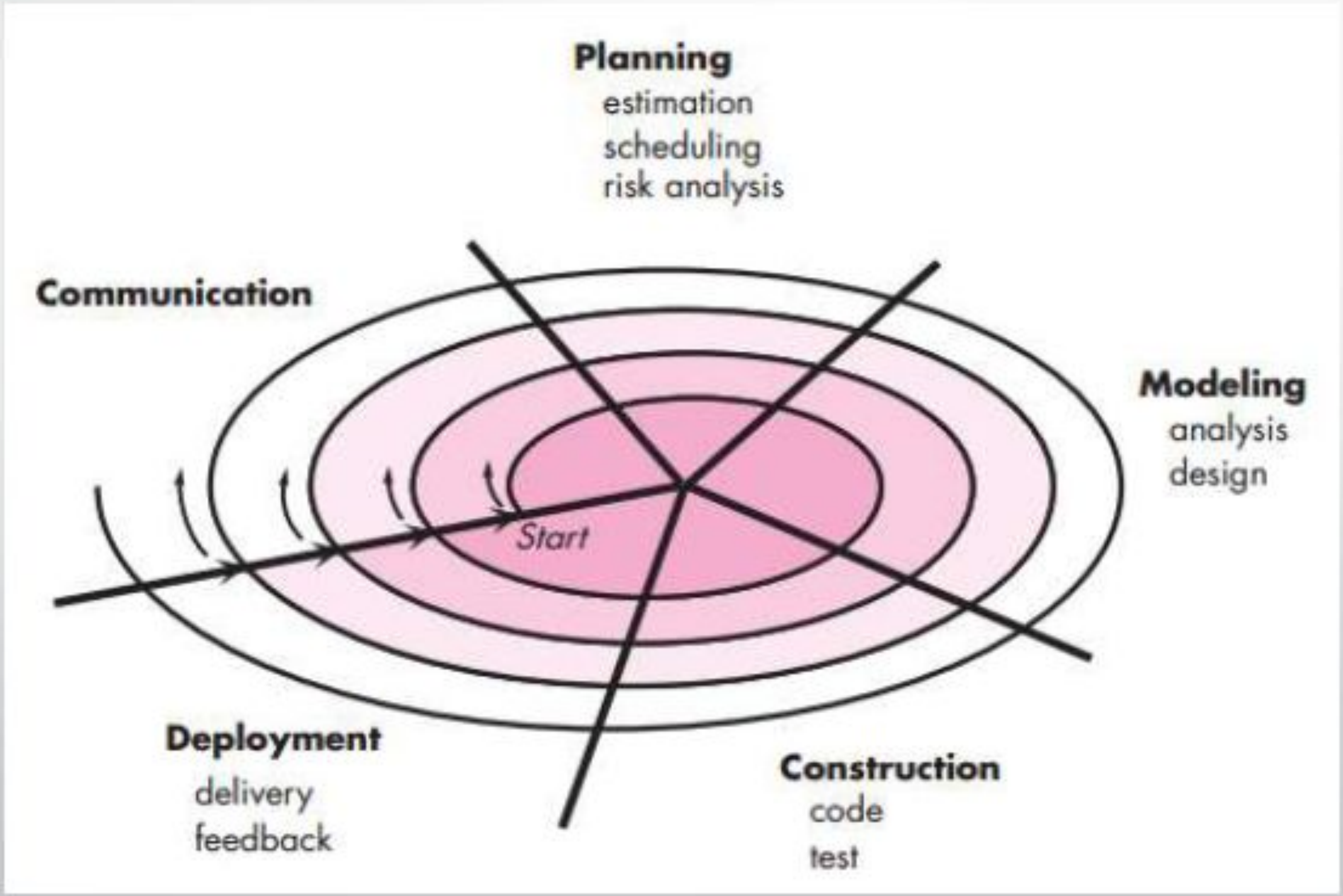
The Prototyping Benefits

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

The Prototyping Problems

- Stakeholders see what appears to be a working version of the software, unaware that the prototype is held together haphazardly, unaware that in the rush to get it working you haven't considered overall software quality or long-term maintainability. When informed that the product must be rebuilt so that high levels of quality can be maintained, stakeholders cry foul and demand that "a few fixes" be applied to make the prototype a working product. Too often, software development management relents.
- As a software engineer, you often make implementation compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability. After a time, you may become comfortable with these choices and forget all the reasons why they were inappropriate. The less-than-ideal choice has now become an integral part of the system

Evolutionary Model: The Spiral



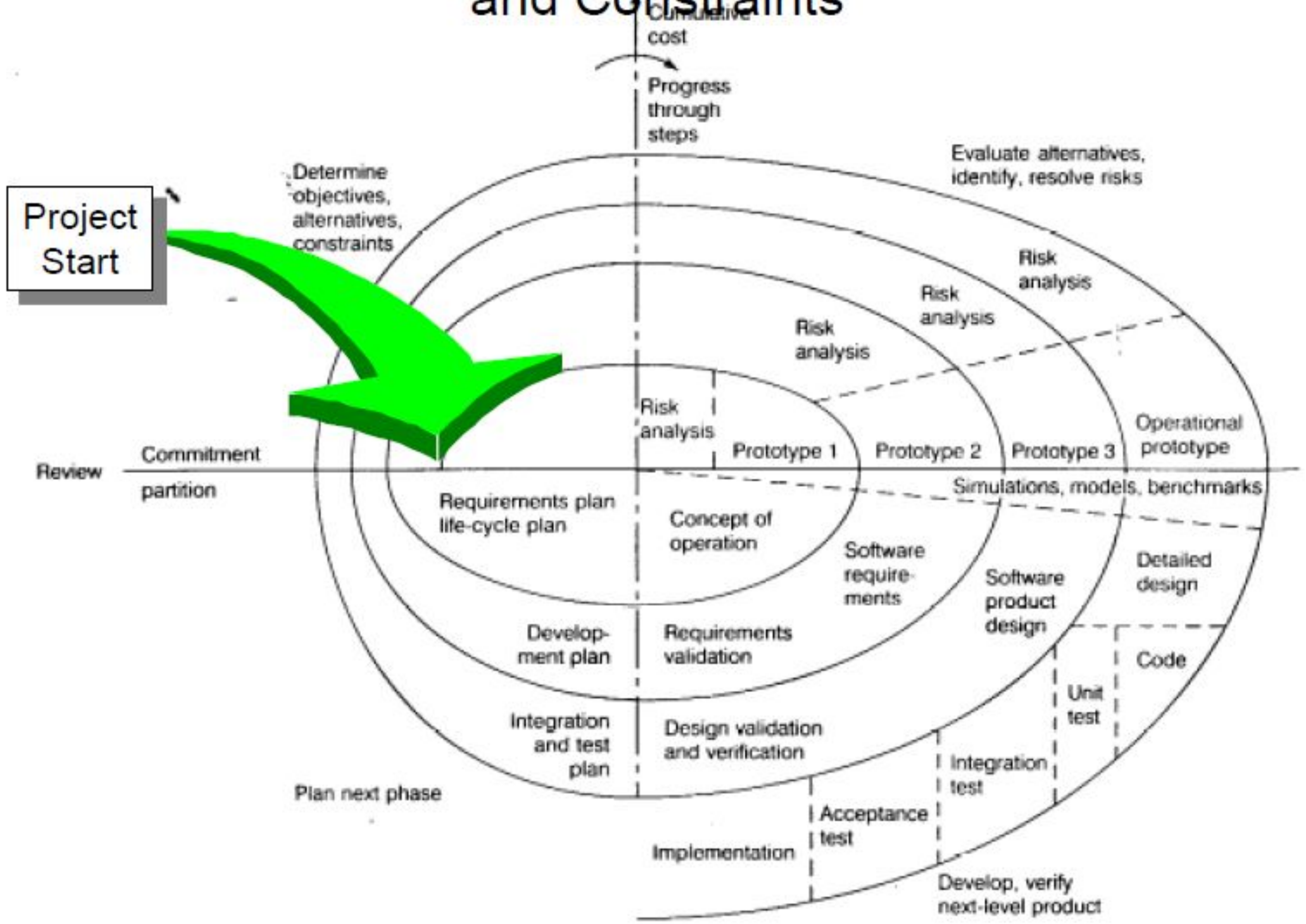
[P

4 Quadrants of spiral model

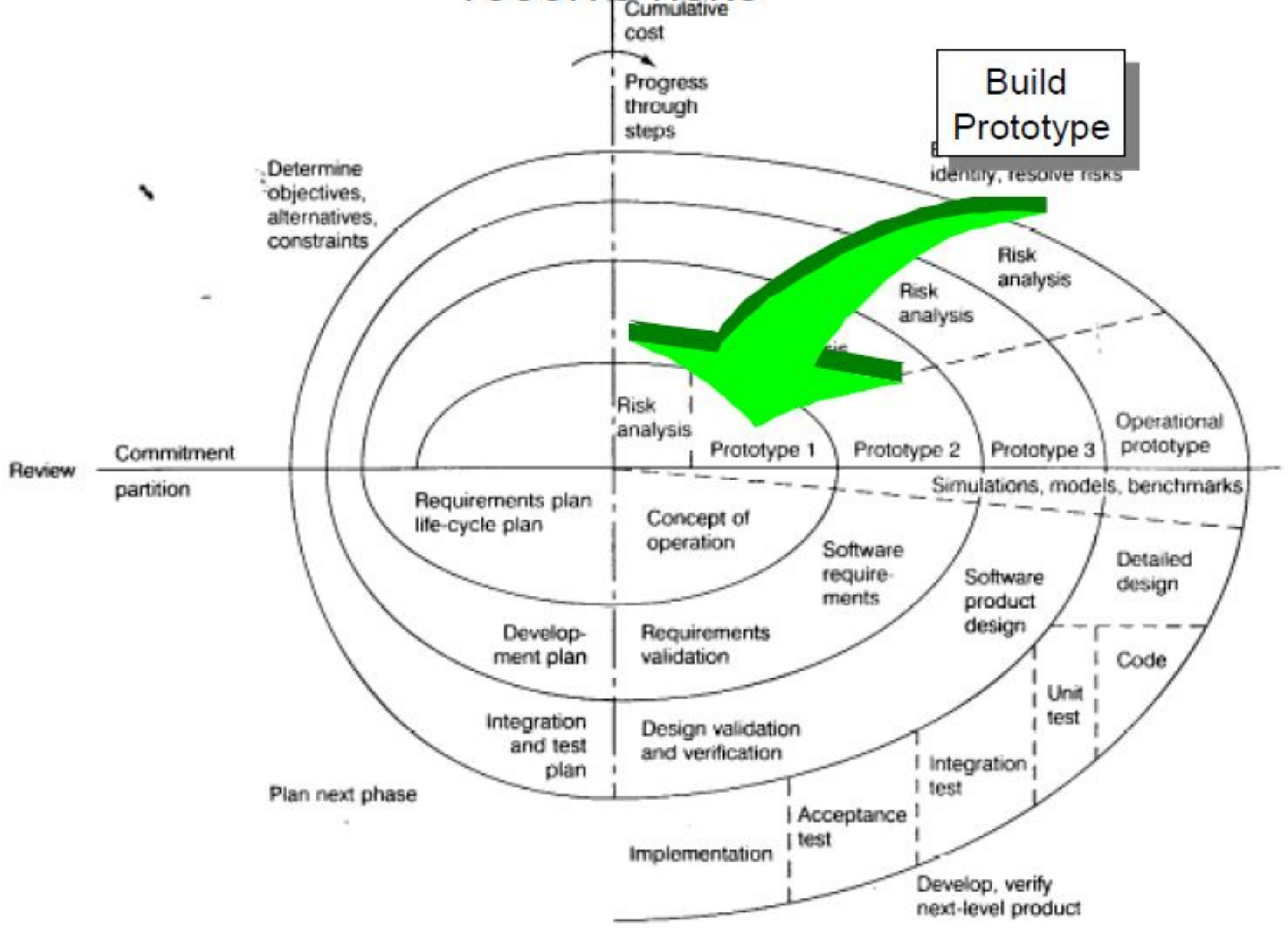
- 1st Quadrant:
  - The objectives are investigated, elaborated and analysed.
  - Risks are also identified.
  - Alternative Solutions are proposed
- 2nd Quadrant:
  - Alternative solutions are evaluated to select best.
- 3rd Quadrant:
  - Developing and verifying the next level of the product
- 4th Quadrant:
  - Reviewing the results of the stages traversed so far with the customer.
  - Planning the next iteration around the spiral.

:

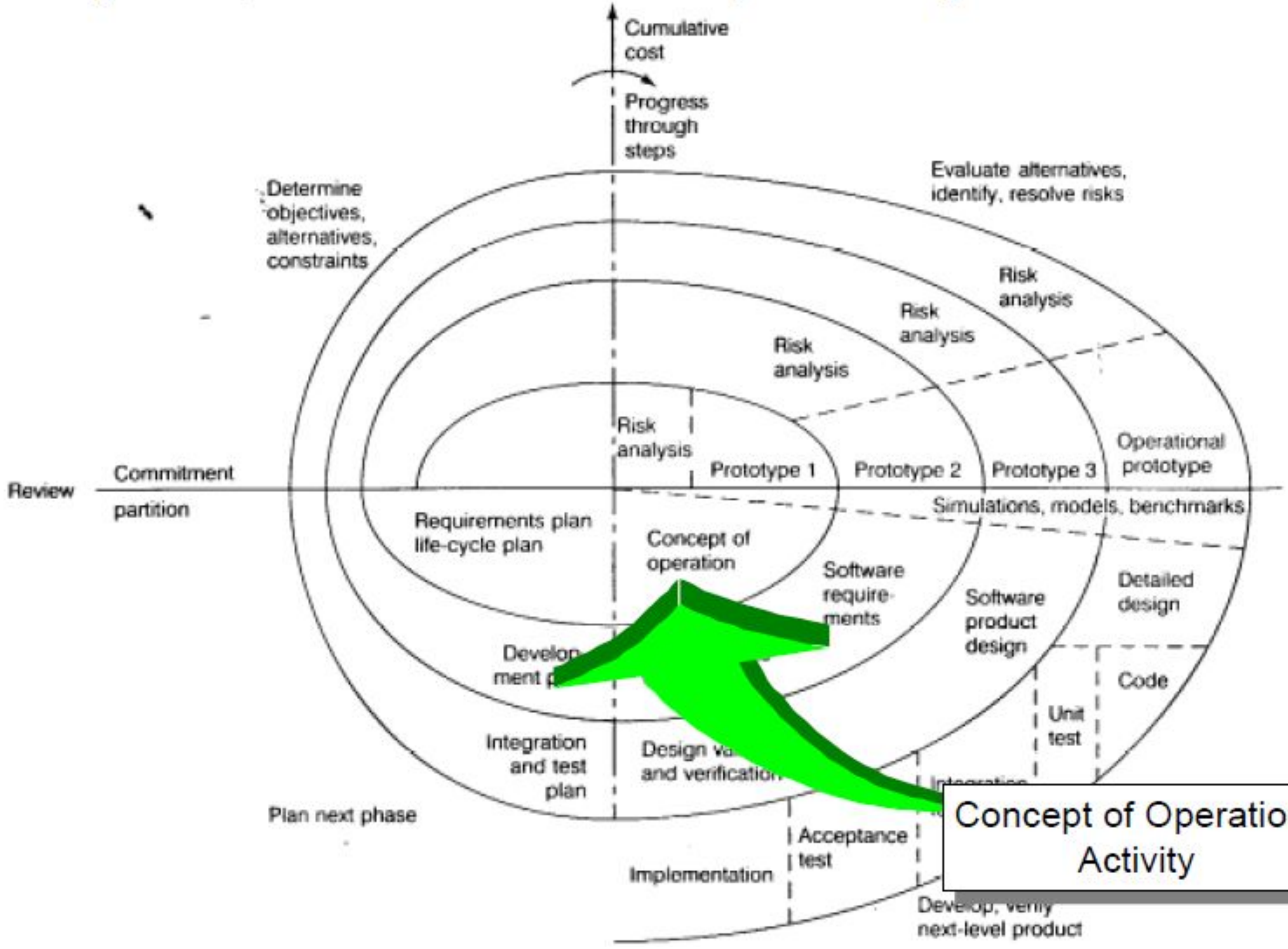
Cycle 1, Quadrant IV: Determine Objectives, Alternatives and Constraints



Cycle 1, Quadrant I: Evaluate Alternatives, Identify, resolve risks

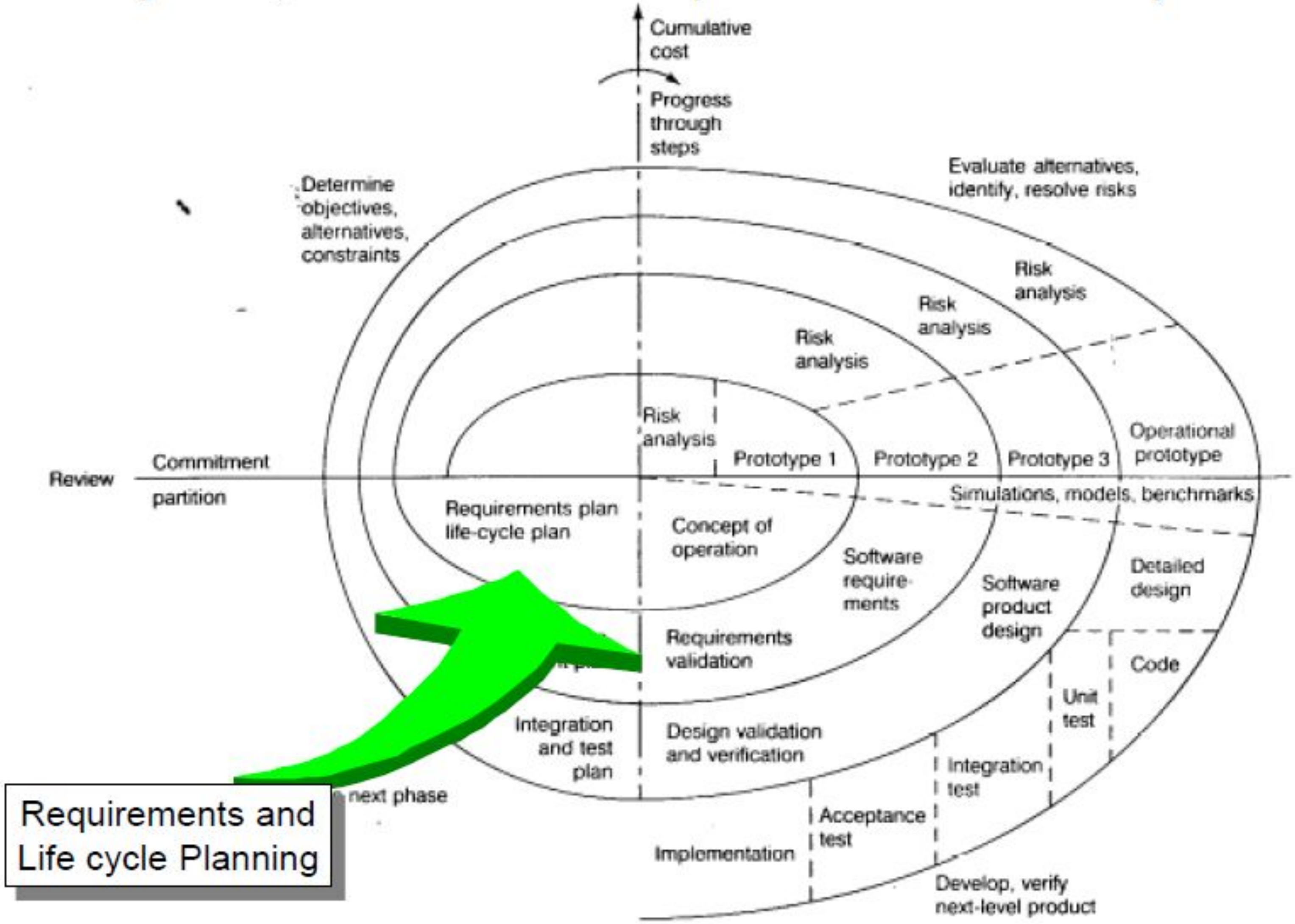


Cycle 1, Quadrant II: Develop & Verify Product



Concept of Operation Activity

Cycle 1, Quadrant III: Prepare for Next Activity



Evolutionary Model: The Spiral

A spiral model is divided into a set of framework activities defined by the software engineering team.

The software team performs activities that are implied by a circuit around the spiral in a clockwise direction, beginning at the center.

Anchor point milestones—a combination of work products and conditions that are attained along the path of the spiral—are noted for each evolutionary pass.

The first circuit around the spiral might result in the development of a product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.

Cost and schedule are adjusted based on feedback derived from the customer after delivery.

Evolutionary Model: The Spiral

ach loop in the spiral is split into four sectors:

Objective setting. Specific objectives for that phase of the project are defined

Risk assessment and reduction. For each of the identified project risks, a detailed analysis is carried out.

Development and validation. After risk evaluation, a development model for the system is chosen.

Planning. The project is reviewed and a decision made whether to continue with a further loop of the spiral.

Selecting Software Models

Table Selections on the Basis of the Project Type and Associated Risks

Project Type and Associated Risks	Waterfall	Prototype	Spiral	RAD	Formal Methods
Reliability requirements	No	No	Yes	No	Yes
Stable funds	Yes	Yes	No	Yes	Yes
Reuse components	No	Yes	Yes	Yes	Yes
Tight project schedule	No	Yes	Yes	Yes	No
Scarcity of resources	No	Yes	Yes	No	No

Selecting Software Models

Table Selection on the Basis of the Requirements of the Project

Requirements of the Project	Waterfall	Prototype	Spiral	RAD	Formal Methods
Requirements are defined early in SDLC	Yes	No	No	Yes	No
Requirements are easily defined and understandable	Yes	No	No	Yes	Yes
Requirements are changed frequently	No	Yes	Yes	No	Yes
Requirements indicate a complex System	No	Yes	Yes	No	No

Selecting Software Models

able Selection on the Basis of the Users

User Involvement	Waterfall	Prototype	Spiral	RAD	Formal Methods
Requires Limited User Involvement	Yes	No	Yes	No	Yes
User participation in all phases	No	Yes	No	Yes	No
No experience of participating in similar projects	No	Yes	Yes	No	Yes

Features	**Original water fall**	**Iterative water fall**	**Prototyping model**	**Spiral model**
Requirement Specification	Beginning	Beginning	Frequently Changed	Beginning
Understanding Requirements	Well Understood	Not Well understood	Not Well understood	Well Understood
Cost	Low	Low	High	Expensive
Availability of reuseable component	No	Yes	yes	yes
Complexity of system	Simple	simple	complex	complex
Risk Analysis	Only at beginning	No Risk Analysis	No Risk Analysis	yes
User Involvement in all phases of SDLC	Only at beginning	Intermediate	High	High
Guarantee of Success	Less	High	Good	High
Overlapping Phases	No overlapping	No Overlapping	Yes Overlapping	Yes Overlapping
Implementation time	long	Less	Less	Depends on project
Flexibility	Rigid	Less Flexible	Highly Flexible	Flexible