# Recap: Programs versus Software Products

| ***Programs*** | ***Software Products*** |
| --- | --- |
| • Usually small in size | • Large in size |
| • Author is sole user | • Multiple users |
| • Single developer | • Team of developers |
| • Lacks proper user interface | • Well-designed interface |
| • Lacks proper documentation | • Well documented & user-manual prepared |
| • Ad hoc development. | • Systematic development |

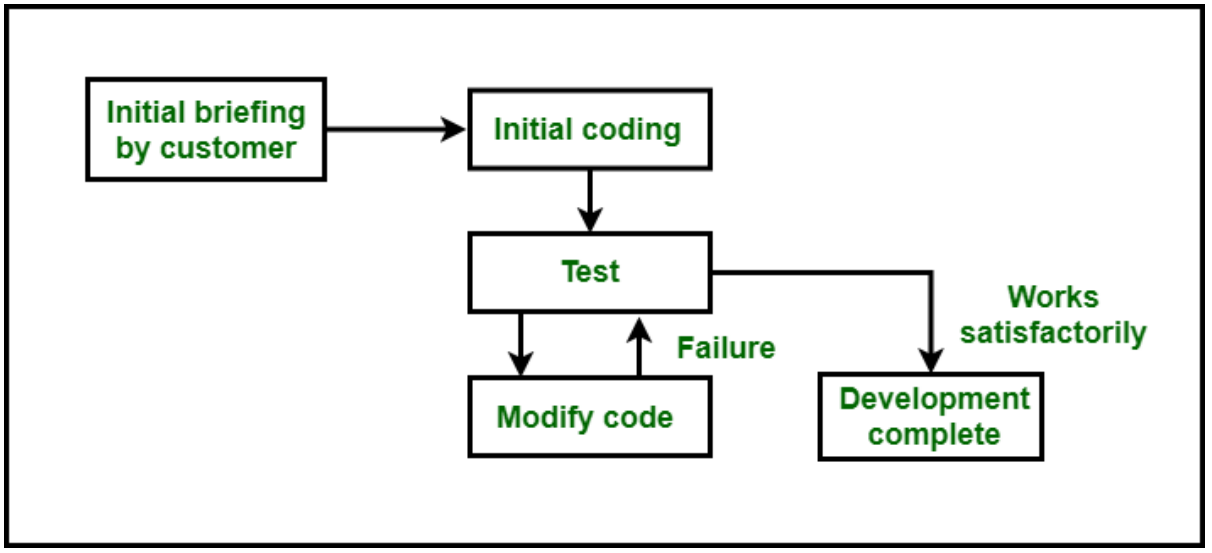
**Program: Single use single developer**

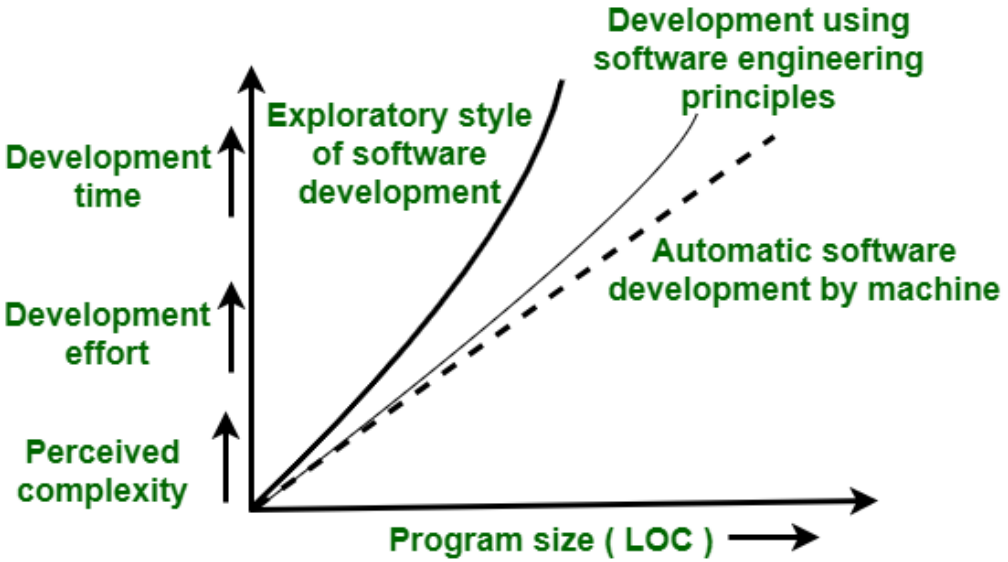
**Software: Multi-user multi-developer**

# Recap: What are the attributes of good software?

- **Efficiency: The software should not make wasteful use of system resources such as memory and processor cycles.**
- **Maintainability: It should be possible to evolve the software to meet the changing requirements of customers. It describes how easily a system can be modified.**
- **Dependability: It is the flexibility of the software that ought to not cause any physical or economic injury within the event of system failure. It includes a range of characteristics such as reliability, security and safety.**
- **In time: Software should be developed well in time.**
- **Within Budget: The software development costs should not overrun and it should be within the budgetary limit.**
- **Functionality: The software should exhibit the proper functionality, i.e, it should perform all the intended task.**
- **Adaptability: The software system should have the ability to get adapted to a reasonable extent with the changing requirements.**
- **and so on….**

# Exploratory style of software development




- It results in unmaintainable code.
- Difficult to work in team.

Why there is exponential growth?

- Long term vs short term memory
- Magical number 7
- Number of variables increases with LOC

Source: https://www.geeksforgeeks.org/introduction-to-exploratory-style-of-software-development/

# Software process

A structured set of activities required to develop a software system.

Software engineering process depends on the types of software system, hence there is no concept of universally applicable software process.
Different companies use different process depending on:

- Software type and environment
- Customer requirement
- Team attributes

Many different software processes exist 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.

These activities are complex activities in themselves, and they include subactivities such as requirements validation, architectural design, and unit testing.

# Software process

Desired Characteristics of Software Process:

- Predictability
- Support Testability and Maintainability
- Support Change
- Early Defect Removal
- Process Improvement and Feedback

For describing processes, we require who is involved, what is produced, and conditions that influence the sequence of activities, that is:

- Products, which are the outcomes of a process activity;
- Roles, which reflect the responsibilities of the people involved in the process;
- Pre- and postconditions, which are statements that are true before and after a process activity has been enacted. For example, precondition: consumer has approved all requirements.

Processes selection depends on capabilities of the software developers in an organization and the characteristics of the systems that needs to be developed. Examples:

- For safety-critical systems, a very structured development process is required where detailed records are maintained.
- For business systems, with rapidly changing requirements, a more flexible, agile process is likely to be better.

# Plan-driven and agile processes

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes. It all depends on ????
- Software process model a.k.a Software Development Life Cycle (SDLC) model is a simplified representation of a software process from a particular perspective, thus it provides partial information. For example, a process activity model shows the activities and their sequence but may not show the roles of the people involved in these activities.

***A software process model is an abstract representation of a process.***

# Software process models

## The waterfall model

Plan-driven model. Separate and distinct phases of specification and development.

## Incremental development

Specification, development and validation are interleaved. May be plan-driven or agile.The system is developed as a series of versions (increments), with each version adding functionality to the previous version.

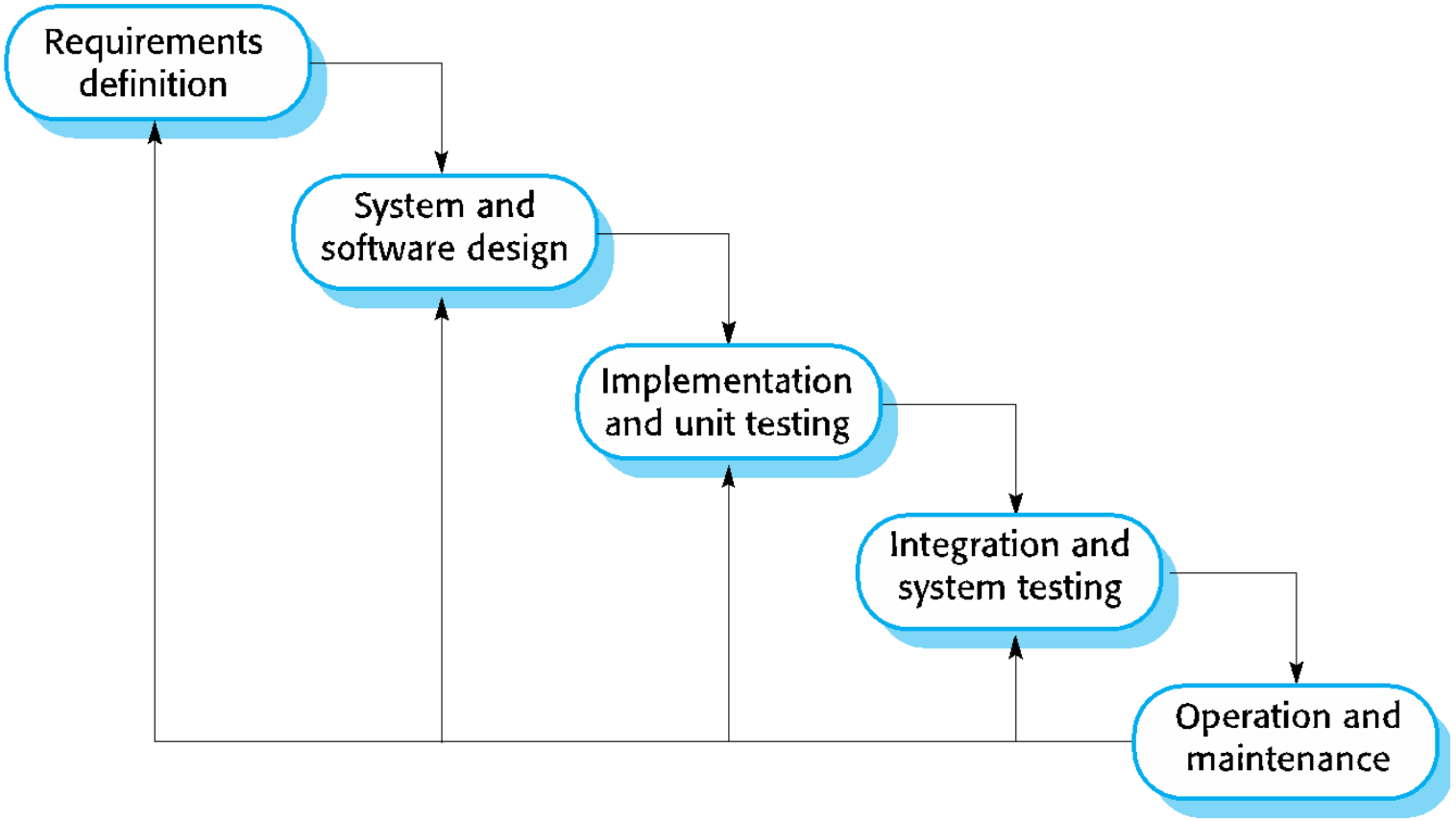
## Integration and configuration

The system development process focuses on configuring the reusable components for use in a new setting and integrating them into a system. May be plan-driven or agile.

In practice, most large systems are developed using a process that incorporates elements from all of these models.

Suggest best suitable process model for safety-critical software and software products

# The waterfall model



# The waterfall model

Principle: You plan and schedule all of the process activities before starting software development.

There are separate identified phases in the waterfall model:

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance

Result of each phase is one or more approved documents (“signed off”).

The following phase should not start until the previous phase has finished. Is it making sense for hardware and software?
The software process, in practice, is never a simple linear model but involves feed-back from one phase to another.

The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway.

# The waterfall model

Unable to accommodate changes, so why to use it?

- *Scenario:* As new information emerges, the documents from the previous stages should be modified and approved from customer to reflect the required system changes. *Problem:* delay in project delivery.
- “Separation of concerns”: Each phase deals with a distinct and separate set of concerns. It breaks complex task of building the software into smaller tasks of specifying requirements, doing design, etc. It provides better handle to the engineers and managers in dealing with the complexity of the problem.

Steps involved in waterfall lifecycle:

- Here, both customers and developers may prematurely freeze the software specification so that no further changes are made to it. This left the problems for later resolution.
- During the final life-cycle phase (operation and maintenance) the software is put into use. Errors in the original software requirements are discovered. Program and design errors emerge, and the need for new functionality is identified.
- The system must therefore evolve to remain useful.
- Making these changes (software maintenance) may involve repeating previous process stages.

In reality, software has to be flexible and accommodate change as it is being developed. But in waterfall model, we follow early commitment and system rework when errors are discovered.

# Uses of the waterfall model

Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.

- This model is useful when the requirements are well-understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.
- Example: automating an existing manual system, embedded systems and critical systems

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.

# The waterfall model problems

- User is not clear about exact descriptions
- Freezing the requirements usually requires choosing the hardware. It is likely that the final software will use a hardware technology on the verge of becoming obsolete.
- “Big bang” approach-the entire software is delivered at the end. This entails heavy risks, as the user does not know until the very end what they are getting. Furthermore, if the project runs out of money in the middle, then there will be no software.
- It encourages “requirements bloating”. Since all requirements must be specified at the start and only what is specified will be delivered, it encourages the users and other stakeholders to add even those features which they think might be needed (which finally may not get used).

# Coping with change

Change is inevitable in all large software projects.

- Business changes lead to new and changed system requirements
- New technologies open up new possibilities for improving implementations
- Changing platforms require application changes

Change leads to rework so the costs of change include both rework (e.g. re-analysing requirements) as well as the costs of implementing new functionality.
How to reduce cost of rework?

Change anticipation, where the software process includes activities that can anticipate possible changes before significant rework is required.

- For example, a prototype system may be developed to show some key features of the system to customers.

Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.

- This normally involves some form of incremental development. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

# Software Prototyping

System prototyping, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This approach supports change anticipation.

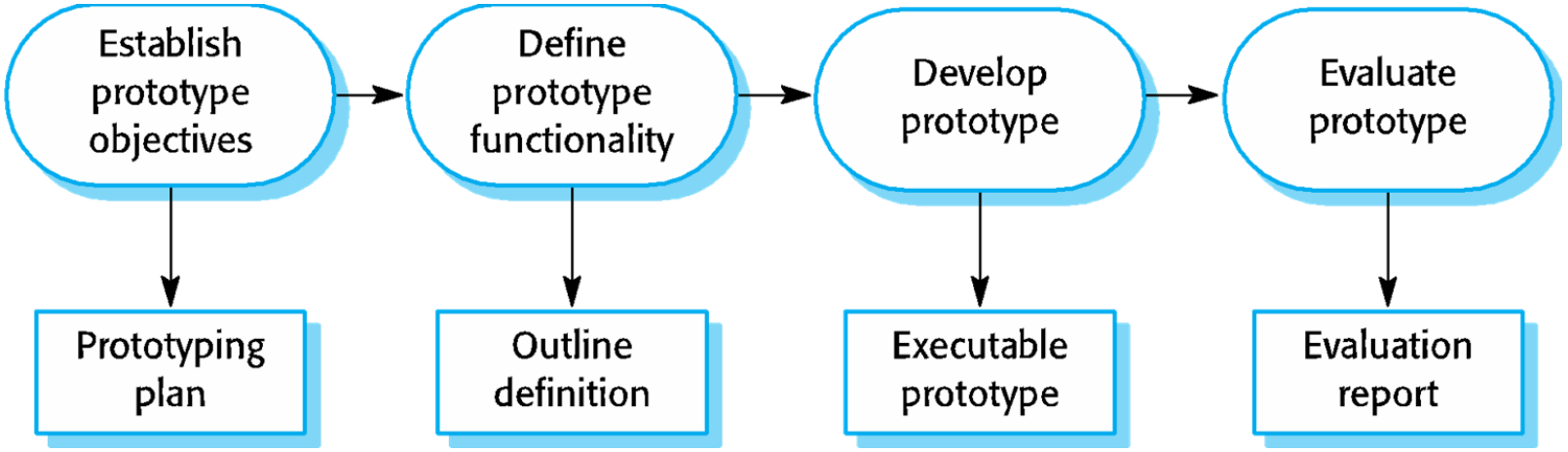
A prototype can be used in:

- The requirements engineering process to help with requirements elicitation and validation;
- In design processes to explore options and develop a UI design;
- In the testing process to run back-to-back tests.

Benefits of prototyping:

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

# Software Prototyping Development



# Software Prototyping Development

May involve leaving out functionality

- Prototype should focus on areas of the product that are not well-understood;
- Error checking and recovery may not be included in the prototype;
- Focus on functional rather than non-functional requirements such as reliability and security

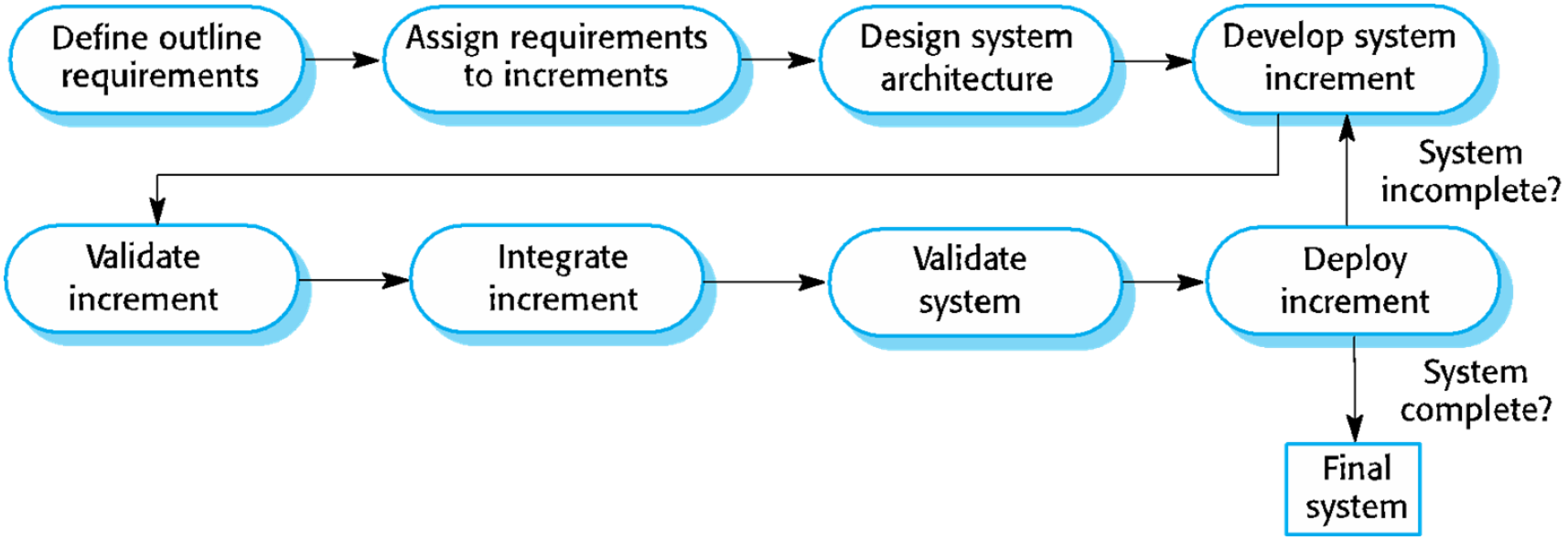
Prototypes should be discarded after development as they are not a good basis for a production system:

- It may be impossible to tune the system to meet non-functional requirements;
- Prototypes are normally undocumented;
- The prototype structure is usually degraded through rapid change;
- The prototype probably will not meet normal organisational quality standards.

# Incremental delivery

- Incremental delivery, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance.
- 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.
- 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.
- Once an increment is completed and delivered, it is installed in the customer’s normal working environment. They can experiment with the system, and this helps them clarify their requirements for later system increments.
- As new increments are completed, they are integrated with existing increments so that system functionality improves with each delivered increment.

# Incremental delivery



# Incremental delivery advantages

Advantages:

- Customer value can be delivered with each increment so system functionality is available earlier.
- 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.

Disadvantages:

- As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- In the incremental approach, there is no complete system specification until the final increment is specified. This requires a new form of contract, which large customers such as government agencies may find difficult to accommodate.
- Iterative delivery is problematic when the new system is intended to replace an existing system. Users need all of the functionality of the old system and are usually unwilling to experiment with an incomplete new system.

# Practically Applying Software process models

**Practical software processes: General model along with features of other models.**

Consider case of large systems engineering and create a software model

- It makes sense to combine some of the best features of all of the general processes.
- You need to have information about the essential system requirements to design a software architecture to support these requirements.
- You cannot develop this incrementally.
- Subsystems within a larger system may be developed using different approaches. Parts of the system that are well understood can be specified and developed using a waterfall-based process or may be bought in as off-the-shelf systems for configuration.
- Other parts of the system, which are difficult to specify in advance, should always be developed using an incremental approach.
- In both cases, software components are likely to be reused.

# Incremental delivery and development

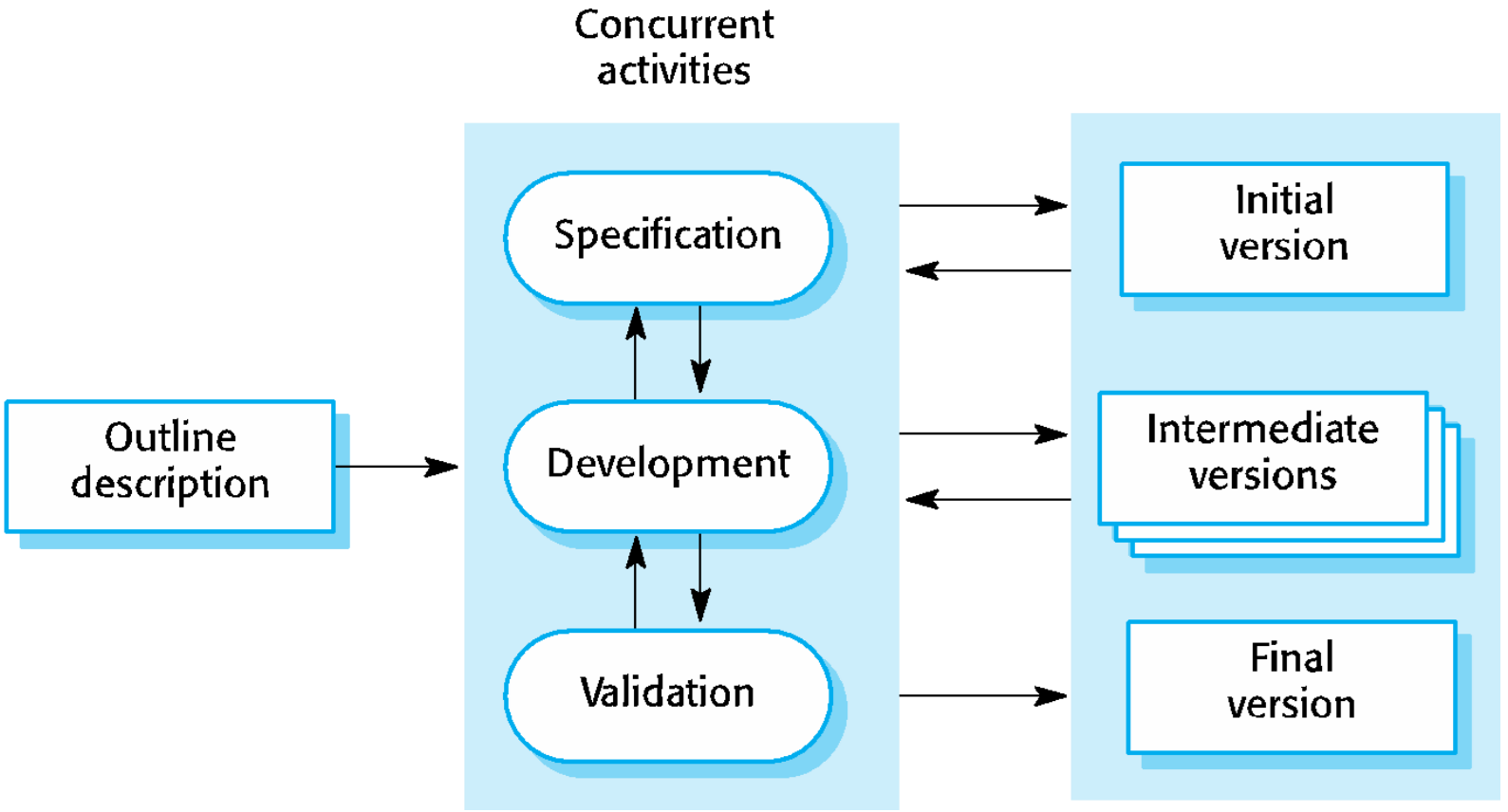
Incremental delivery

- Deploy an increment for use by end-users;
- More realistic evaluation about practical use of software;
- Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

Incremental development

- Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
- Normal approach used in agile methods;
- Evaluation done by user/customer proxy.

# Incremental development




*Intuition*: Develop an initial implementation, getting feedback from users and others, and evolving the software through several versions until the required system has been developed

- In plan-driven approach, the system increments are identified in advance.
- In agile approach, the early increments are identified, but later increments depends on progress and customer priorities.

# Incremental development

It uses incremental development, i.e., develop a system incrementally and expose it to customers for comment, without necessarily delivering it and deploying it in the customer's environment

Each increment or version of the system incorporates some of the functionality that is needed by the customer.
Generally, early increments are the most important or most urgently required functionality.

It is problematic for complex systems where different teams develop different parts of the system. They need responsibilities of the different teams working on parts of the system need to be clearly defined with respect to that architecture.

Incremental development Vs Incremental delivery

- Incremental delivery means that the software is used in real, operational processes, so user feedback is likely to be realistic.

Each entry in the list is a task that should be performed in one step of the iterative enhancement process and should be simple enough to be completely understood. Selecting tasks in this manner will minimize the chances of error and reduce the redesign work.

# Incremental development

## 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.

## Problems

The process is not visible.

- Managers need regular deliverables to measure progress. 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.

# Modified Incremental delivery

One common approach for iterative development:

- Perform requirement specifications and the architecture design in a standard waterfall or prototyping approach, i.e., most of the requirements are specified upfront.
- Then, deliver the software iteratively. At the start of each delivery iteration, which requirements will be implemented in this release are decided, and then the design is enhanced and code developed to implement the requirements.
- The iteration ends with delivery of a working software system.

How to select requirements in an iteration?

- Depends on the value the requirement provides to the end users and how critical they are for supporting other requirements.

Advantages:

- Requirements are mostly known: waterfall advantages.
- Rework in development iterations will diminish: iterative approach advantages.
- It does not have the all-or-nothing risk.

What problems this approach can have?

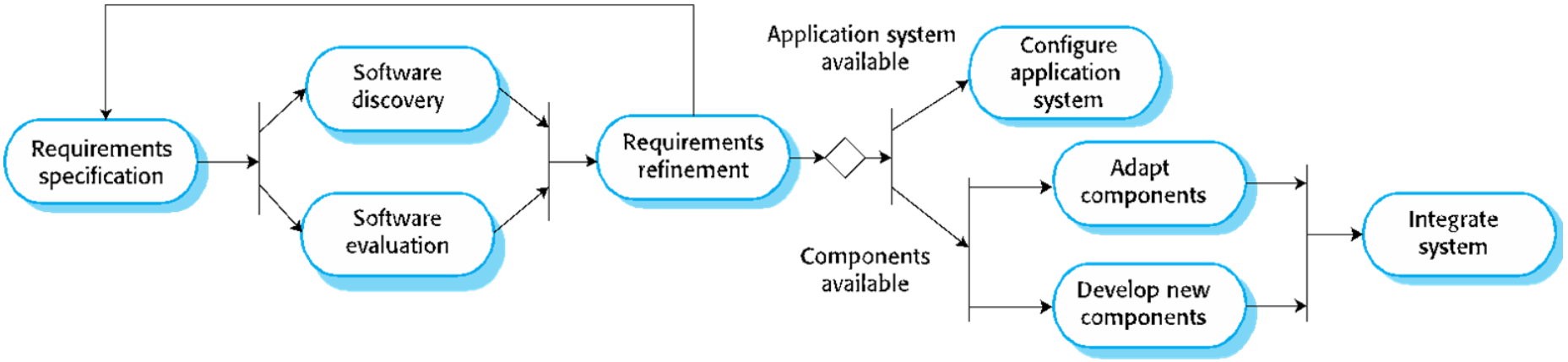
# Integration and configuration

- Based on software reuse where systems are integrated from existing components or application systems (sometimes called COTS -Commercial-off-the-shelf) systems).
- Reused elements may be configured to adapt their behavior and functionality to a user's requirements
- Reuse is now the standard approach for building many types of business system
- Example of reusable software: Stand-alone application systems (sometimes called COTS) that are configured for use in a particular environment.

## Advantages and disadvantages

- Reduced costs and risks as less software is developed from scratch
- Faster delivery and deployment of system
- But requirements compromises are inevitable so system may not meet real needs of users
- Loss of control over evolution of reused system elements

# Reuse-oriented software engineering




- **Requirements specification**: Brief descriptions of essential requirements and desirable system features.
- **Software discovery and evaluation**: Using outline of the software requirements, a search is made for components and systems that provide those functionality. Candidate components and systems are evaluated to see if they meet the essential requirements and used in the system.
- **Requirements refinement**: Requirements are refined using information about the reusable components. The requirements are modified to reflect the available components, and the system specification is re-defined.
- **Application system configuration**: Configuring off-the- shelf application system to create the new system.
- **Component adaptation and integration**: If there is no off-the-shelf system, individual reusable components may be modified and new components developed. These are then integrated to create the system.

# Timeboxing Model

How to reduce the average delivery time for iterations ?

- Parallelism between the different iterations, i.e., start new iteration before current iteration is finished.

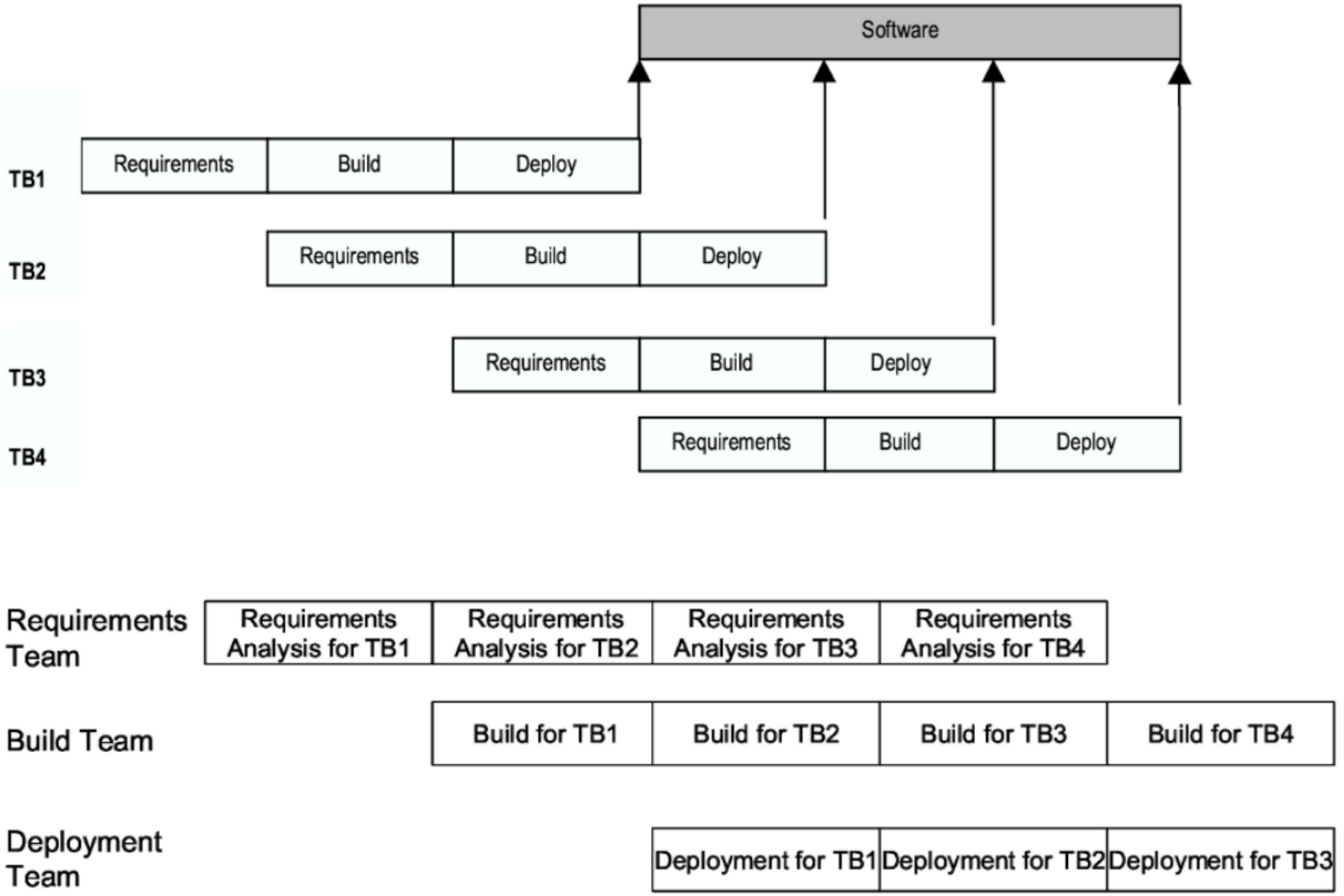
*Requirement: Proper structuring of each iteration and teams co-ordination.*

It's basic unit is a time box, which is of fixed duration.

*Requirements or features will be built in a time box so they should be properly selected. In other iterative approaches, the functionality is selected and then the time to deliver is determined. This perspective of development, makes the schedule a nonnegotiable and a high-priority commitment.*

- Each time box is divided into a sequence of stages, like waterfall model.
- Each stage performs some clearly defined task for the iteration.
- Duration of each stage is approximately the same.
- *There is a dedicated team for each stage.*

# Timeboxing Model




Tasks of different teams.

Tradeoff to achieve reduced time:

- Increase resources and project management
- Timeboxing requires dedicated teams for different stages and the total size is their sum.
- Other models use single team to perform nearly all the stages.

It is suited for projects that require a large number of features to be developed in a short time around a stable architecture using stable technologies.

# Rational Unified Process (RUP)

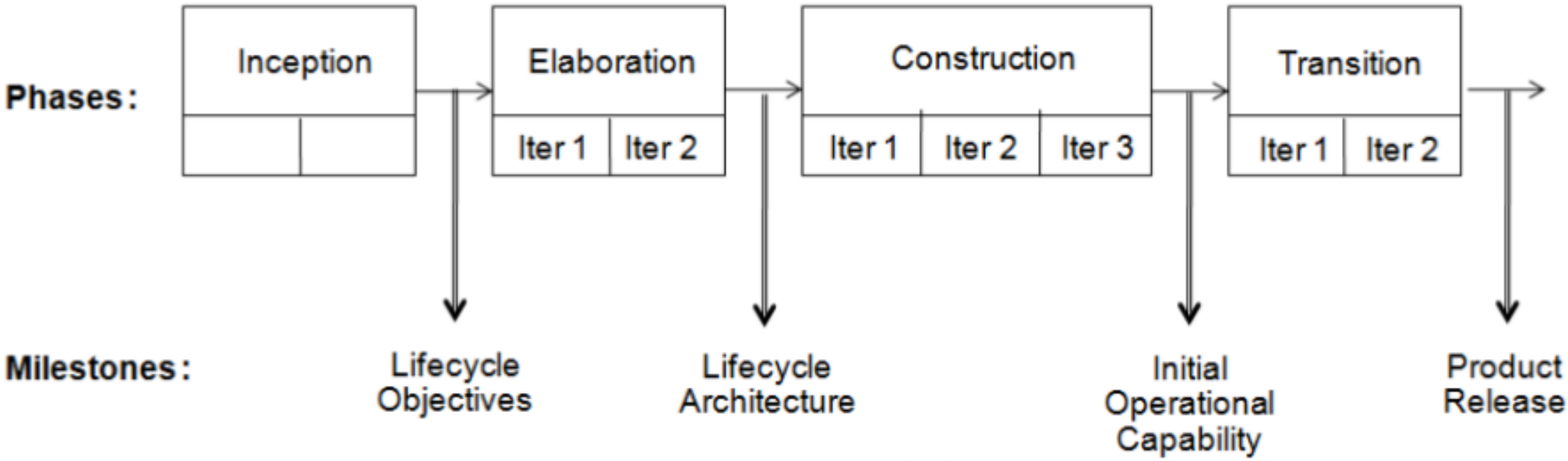
Iterative process model by Rational, now part of IBM.
Software development is accomplished into cycles. Each cycle delivers a fully working system with some additional capability built on an existing system (built in the previous cycle).

Each cycle consists of following four consecutive phases:

- Inception phase: Specify the vision and high-level capability, what business benefits it is expected to provide and a basic plan of the project regarding the cost and schedule.
- Elaboration phase: Requirements are specified, high-level project plan is prepared, critical engineering decisions ( technologies and architecture, etc.) are taken, and a detailed understanding of the project exists.
- Construction phase: Software is built and tested.
- Transition phase: Moving software from the development environment to the client's environment. It may require additional testing, conversion of old data for this software to work, training of personnel, etc.

# Rational Unified Process (RUP)

Phases are consecutive but they may have multiple iterations. For example: In the elaboration phase, the first iteration may just specify the overall architecture and high-level requirements, while the second iteration may be done to thrash out the details.
There may be multiple iterations to transition the developed software, with each iteration “making live” some part or some feature of the developed software.




The RUP model.

# RUP Vs Other models

RUP has separated the phases from the tasks and allows multiple of these subprocesses to function within a phase.

So, a project, if it so wishes, may do detailed requirements only for some features during the elaboration phase, and may do detailing of other requirements while the elaboration is going on.

It provides better flexibility in planning because it is often not possible to specify all requirements at the start and it is best to start the project with some requirements and work out the details later.

RUP provides a flexible process model, which follows an iterative approach not only at a top level (through cycles), but also encourages iterative approach during each of the phases in a cycle.

Activity level of subprocesses in different phases of RUP.

| | **Inception** | **Elaboration** | **Construction** | **Transition** |
|---|---|---|---|---|
| **Requirements** | High | High | Low | Nil |
| **Anal. and Design** | Low | High | Medium | Nil |
| **Implementation** | Nil | Low | High | Low |
| **Test** | Nil | Low | High | Medium |
| **Deployment** | Nil | Nil | Medium | High |
| **Proj. Mgmt.** | Medium | Medium | Medium | Medium |
| **Config. Mgmt** | Low | Low | High | High |

Thanks

# Process activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- The four basic process activities of specification, development, validation and evolution are organized differently in different development processes.
- For example, in the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.
- These activities are carried out depends on the type of software being developed, the experience and competence of the developers, and the type of organization developing the software.

# Software specification/ requirements engineering

The process of establishing what services are required and the constraints on the system's operation and development.

Mistakes in this stage lead to later problems in the system design and implementation. *Need Rectification:WHY????*

Requirements are presented at two levels of detail:

1) End-users and customers need a high-level statement of the requirements
2) System developers need a more detailed system specification.

## Requirements engineering process

1) Requirements elicitation and analysis

✓ What do the system stakeholders require or expect from the system?
✓ May include prototype design

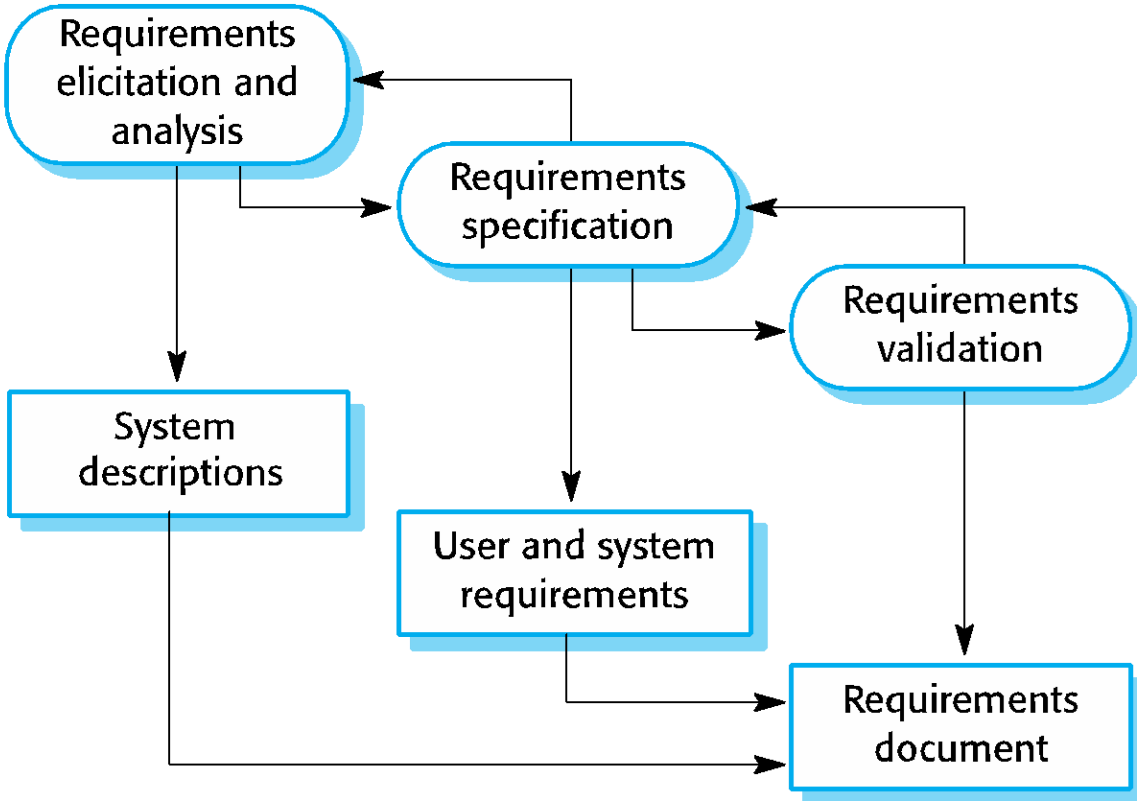
2) Requirements specification

✓ Defining and documenting the requirements in detail

3) Requirements validation

✓ Checking the validity, consistency, and completeness of the requirements
✓ Errors in the requirements document are modified

# Software specification/ requirements engineering




Why these stages are interleaved?

# Software design and implementation

The process of converting the system specification into an executable system.

Software design

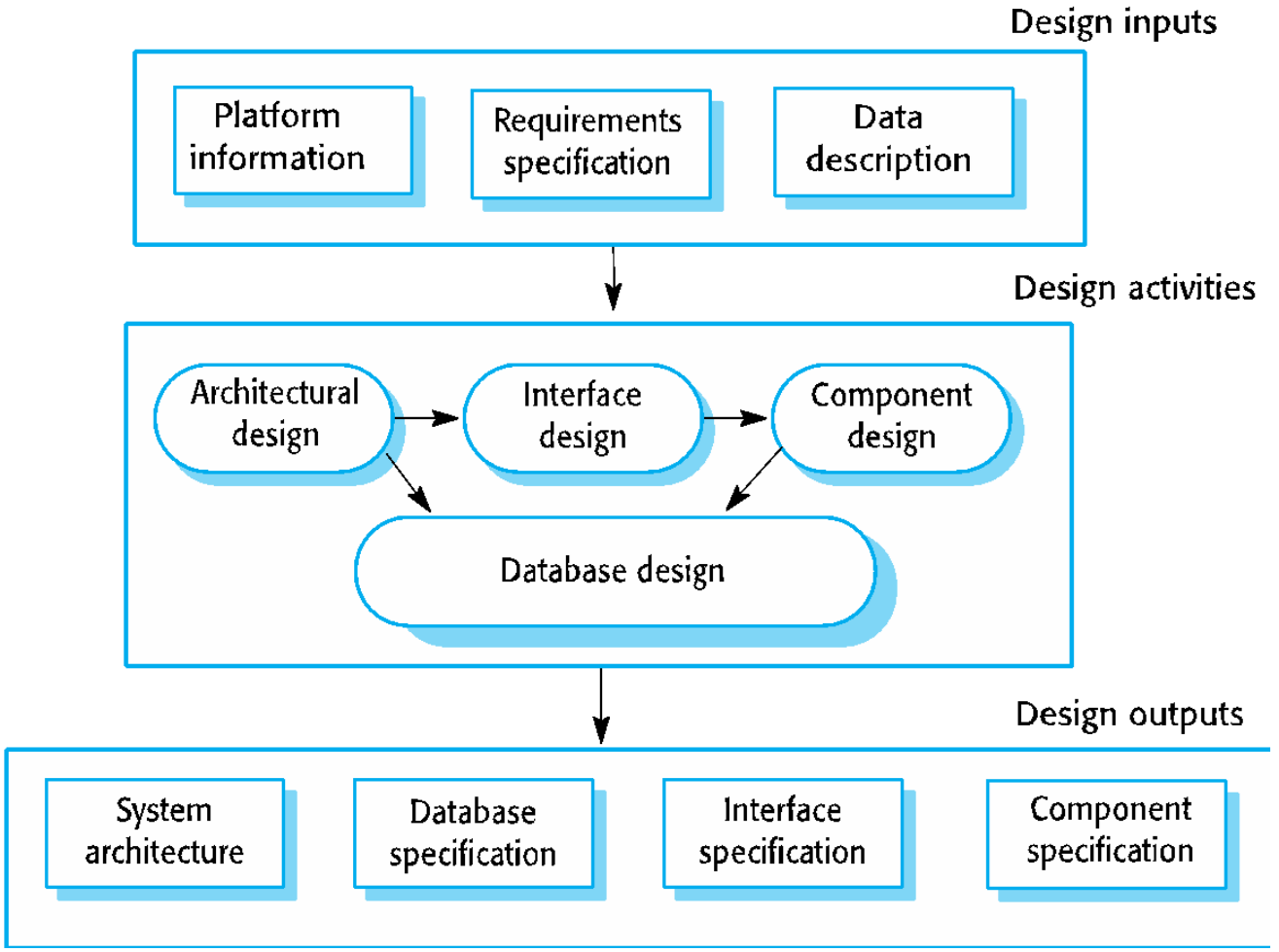
- ✓ Design a software structure that realises the specification;
- ✓ Consists of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used.
- ✓ Designers do not arrive at a finished design immediately but develop the design in stages. They add detail as they develop their design, with constant backtracking to modify earlier designs.

Implementation

- ✓ Translate this structure into an executable program;

The activities of design and implementation are closely related and may be inter-leaved.

# A general model of the design process




- It is both interleaved and interdependent.
- New information about the design is constantly being generated, and this affects previous design decisions. Design rework is therefore inevitable.

# Design activities

- *Architectural design*, where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.
- *Database design*, where you design the system data structures and how these are to be represented in a database.
- *Interface design*, where you define the interfaces between system components. (Component may be used by other components without knowing how it is implemented.)
- *Component selection and design*, where you search for reusable components. If unavailable, you design how it will operate.

These activities depends on the type of system being developed. For example, real-time systems require an additional stage of timing design but may not require database design.

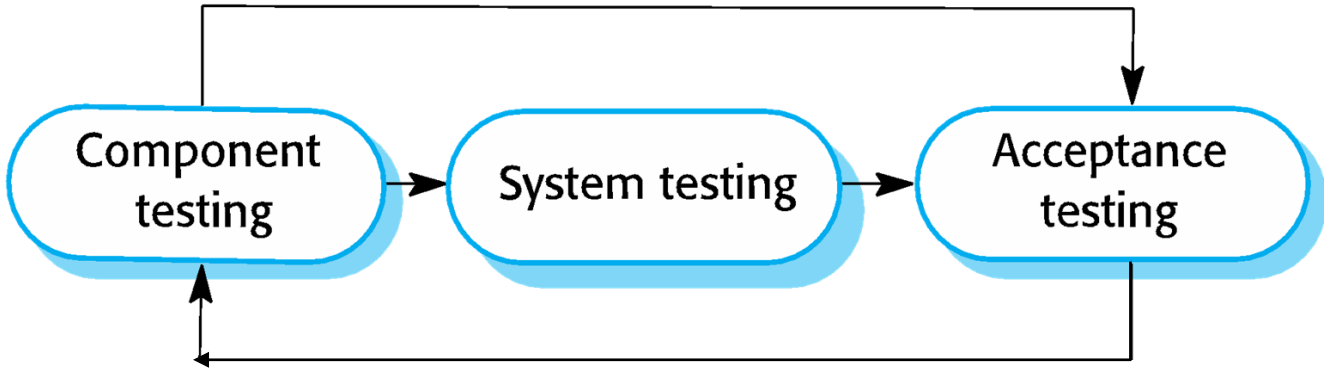
# System implementation

- The software is implemented either by developing a program or programs or by configuring an application system.
- Design and implementation are interleaved activities for most types of software system.
- But for safety-critical systems, we usually design in detail before any implementation begins.
- Programming is an individual activity with no standard process.
- Debugging is the activity of finding program faults and correcting these faults.

# Software validation

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and review processes and system testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- Testing is the most commonly used V & V activity.

# Stages of testing




When defects appear the program is debugged, and this may require testing iteratively. Errors in components, may come during system testing.

## Component testing

- ✓ Individual components are tested independently to find component defects;
- ✓ Components may be functions, objects or coherent groupings of these.

## System testing

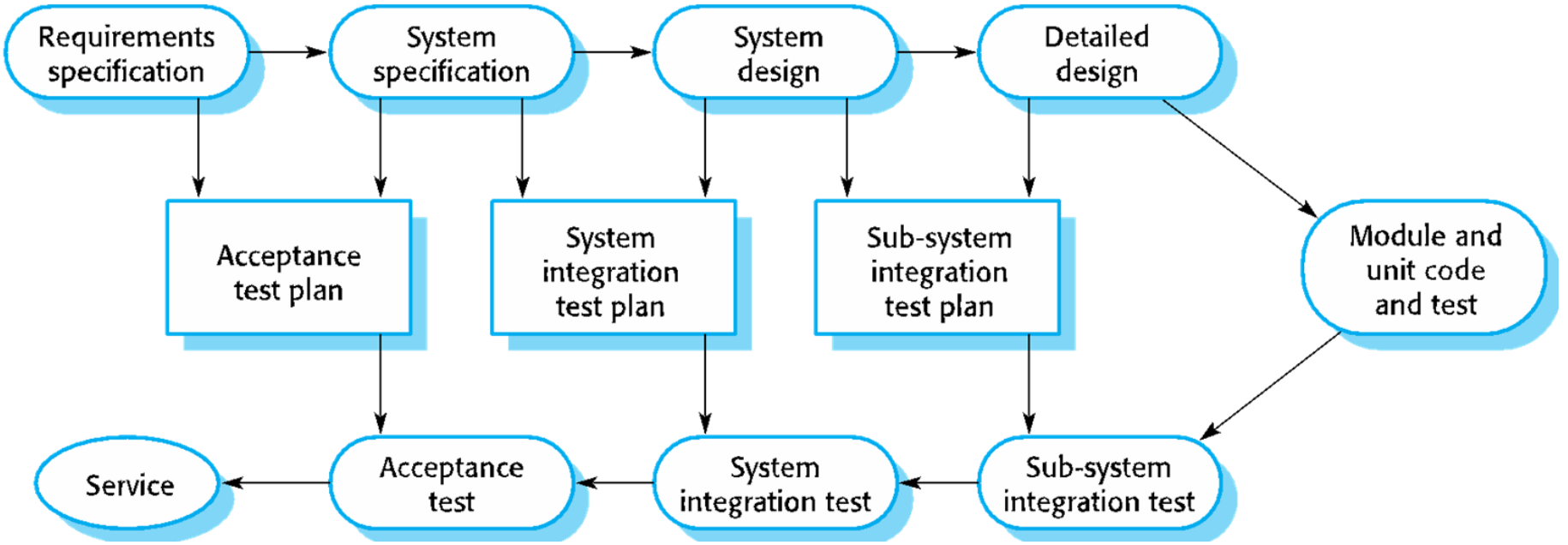
- ✓ Testing of the system as a whole to find interface and integration problems. Testing of emergent properties is particularly important.

## Customer testing

- ✓ Testing with customer data to check that system meets the customer's needs.

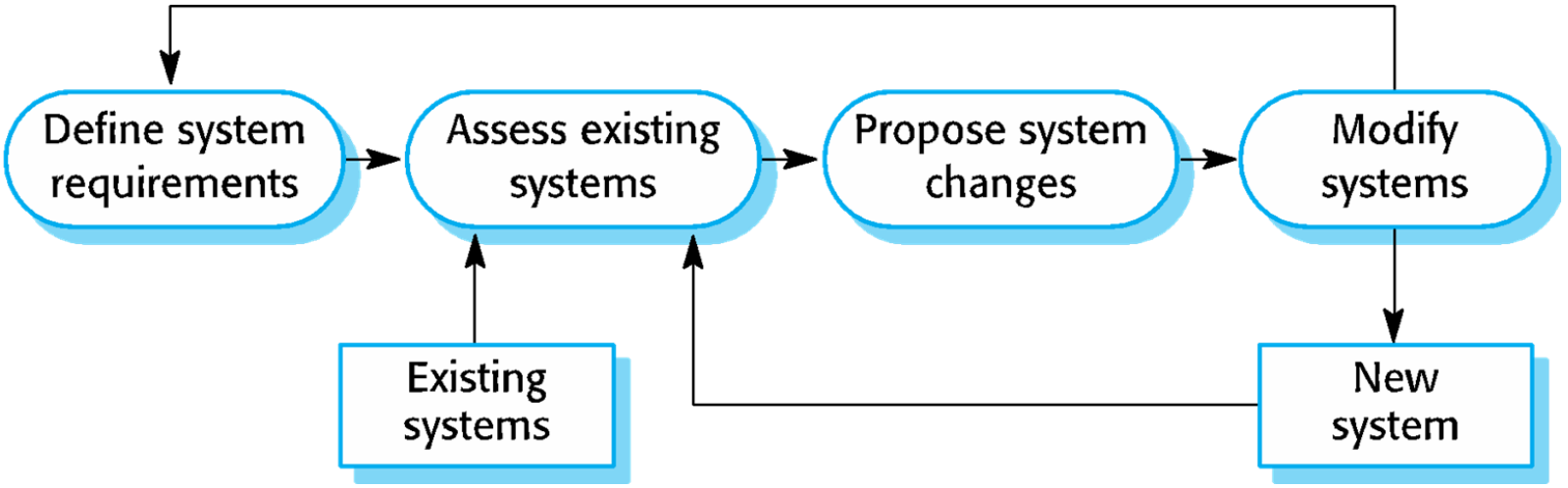
*Beta testing* involves delivering a system to a number of potential customers before delievery who agree to use that system. They exposes the product to real use and detects errors that may not have been anticipated by the product developers. From their feedback, the software product may be modified and released for further beta testing or general sale.

# Testing phases in a plan-driven software process (V-model)



# Software evolution

- As requirements change due to the changing business circumstances, the software that supports the business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.



Thanks