PROCESS MODELS

The software engineering approaches emphasize software development through a well-defined and ordered set of activities.

These activities are graphically modelled (represented) as well as textually described and are variously called as software life cycle model, software development life cycle (SDLC) model, and software development process model.

PREDICTIVE AND ADAPTIVE DEVELOPMENT

PREDICTIVE DEVELOPMENT MODEL

- Predict in advance **what needs to be done.**
- Based on past experience, it is easy to predict the time to build.
- It's often hard to predict exactly what a software application needs to do and how we build it ahead of time.
- Sometimes, may not be familiar with the new programming tool.
- In changing business situations , customer's needs also changes as time goes.

ADAPTIVE DEVELOPMENT MODEL

- Enables you to change the project's goals if necessary during the development.
- Periodically revaluate and decide whether need to change the direction.
- Even then, we cannot say that an adaptive model is always better than a predictive one.
- For example: predictive models work well when the project is relatively small; we know exactly what you need to do, and the timescale is short enough that the requirement won't change during development.

Advantages of predictive model

- **Predictability** – If everything goes according to plan, we can know exactly when different stages will occur and when we will be finished.
- **Stability** – customers know exactly what they are getting.
- **Cost-savings** – if the design is clear and correct, we won't waste time.
- **Detailed design** – if we design everything up front, we shouldn't waste time making a lot of decisions during devlopment.

- **Less refactoring** – Adaptive projects tend to require refactoring.
- A programmer writes some code, sometimes later, the requirements change and the code needs to be modified. These problems don’t occur as often in predictive projects.
- **Fix bugs early** – If the requirements and design are correct and complete, we can fix the bugs they would have caused later.

- **Better documentation** – Predictive models require a lot of documentation. This documentation helps the new people to understand the projects in a better way.

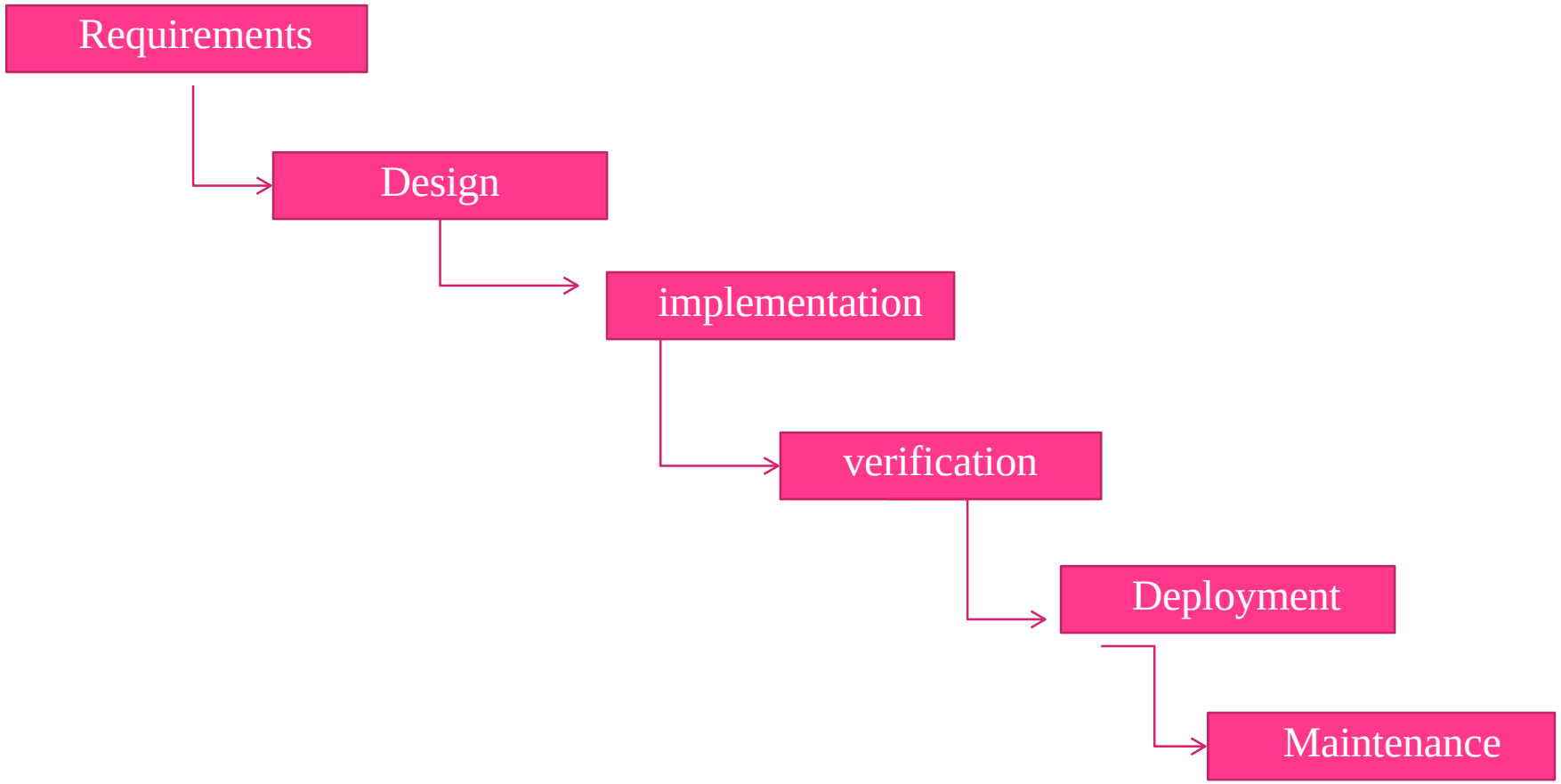
- **Easy maintenance** – in the predictive model we can create a more elegant design that's more consistent and easier to maintain.

Disadvantages of predictive model

- **Inflexible:** If the requirement change, it is very hard to implement.

- **Later initial release:** Many adaptive models enables us to give the users a program as soon as it does something useful. With a predictive model, the users don't get anything until development is finished.

WATERFALL MODEL



WATERFALL MODEL

- Predictive model
- Finish each step completely and thoroughly before move on to the next step.

- The waterfall analogy can be explained like this:

- The water represents information and the stages act like buckets.
- When one bucket is full, the information floods from that bucket into the next so that it can direct the following task.

- **The model can work reasonable well if all the following assumptions are satisfied:**

  - The requirements are precisely known in advance
  - The requirements include no unresolved high risk items.
  - The requirements won't change much during development
  - The team has previous experience with similar projects so that they know what is involved in building the application.
  - There is enough time to do everything sequentially.

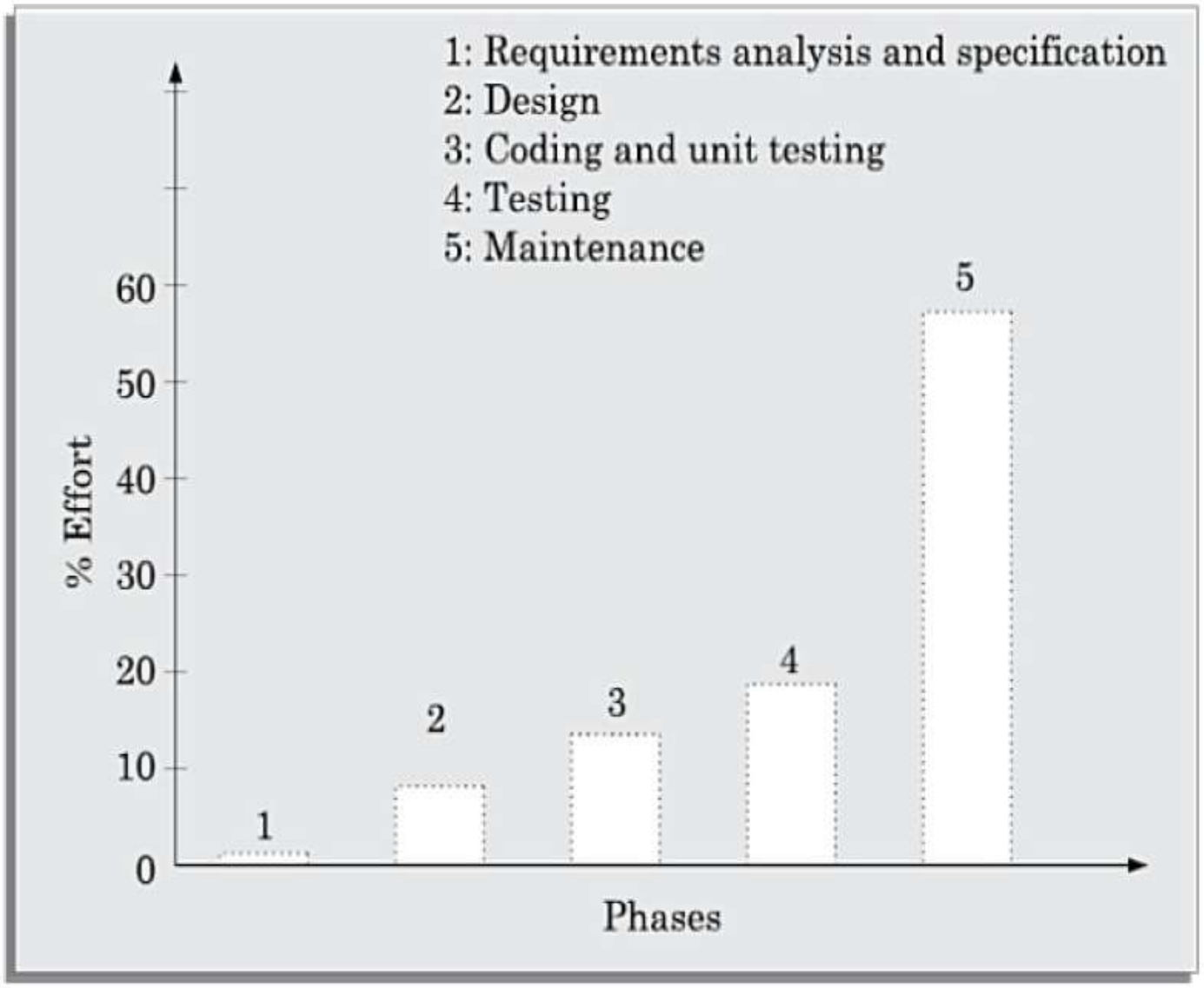
Contd…

- we can add additional steps or split steps to give more details if we like.
- Eg: we can split the Testing into Verification and Validation.

- we can also elaborate on a step .

  Eg: break design into user interface design, high level design,low level design,etc

Relative effort distribution among different phases of a typical product.



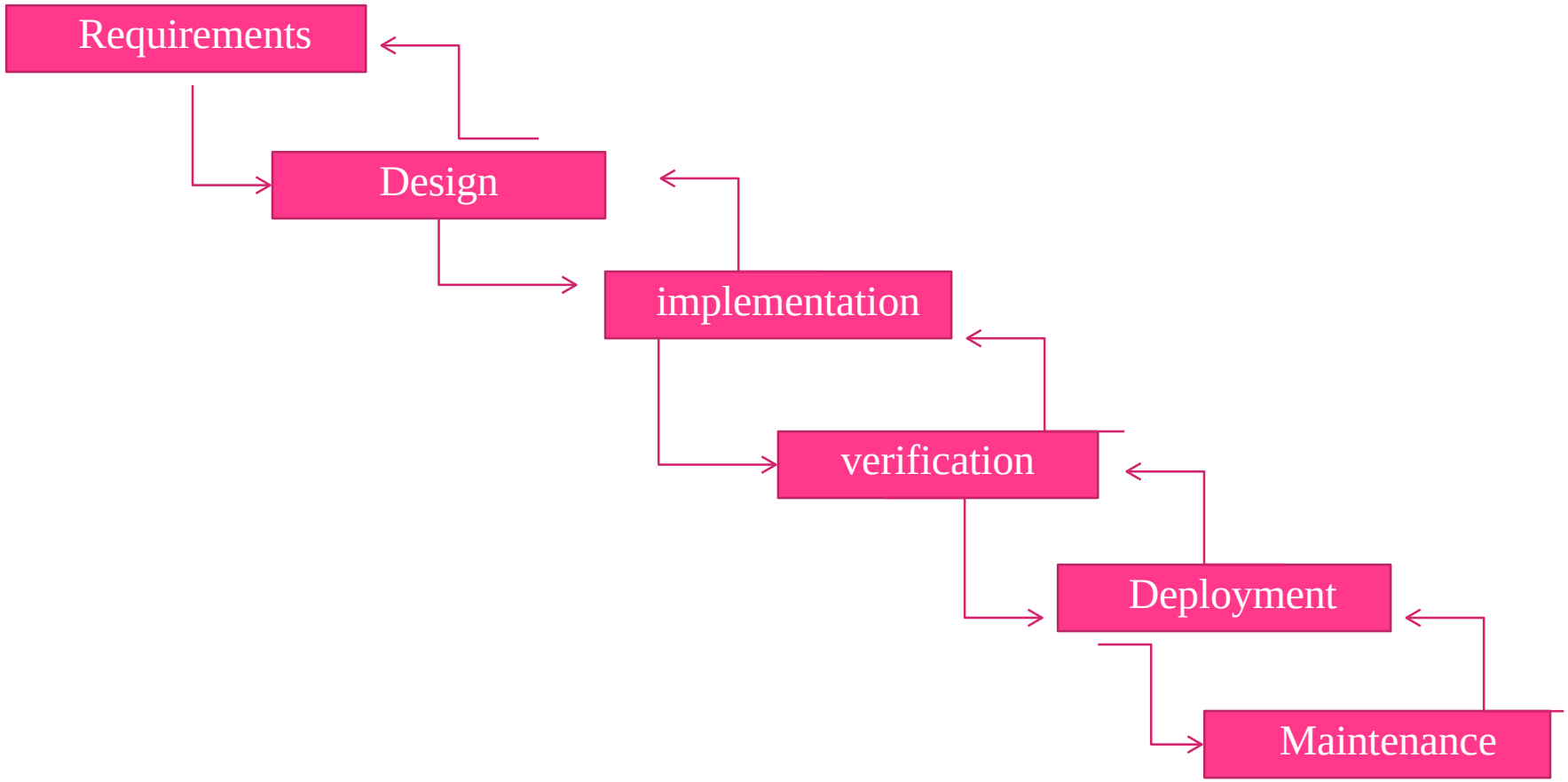
Waterfall Strengths

- **Easy to understand**, easy to use
- **Provides structure** to inexperienced staff
- **Milestones are well understood**
- Sets requirements **stability**
- Good for **management control** (plan, staff, track)

Waterfall Deficiencies

- All **requirements must be known** upfront
- Does not reflect **problem-solving nature of software** development – iterations of phases
- **Integration** is one big bang at the end
- **Little opportunity for customer** to preview the system (until it may be too late)

WATERFALL MODEL with feedback



Contd…

- Sometimes, to go back to the previous stage is quite harder.
- For example, if we are working on implementation and discover a problem in the requirements, it's hard to skip back up two stages to fix the problem.
- It's also less meaningful to move back the previous steps.

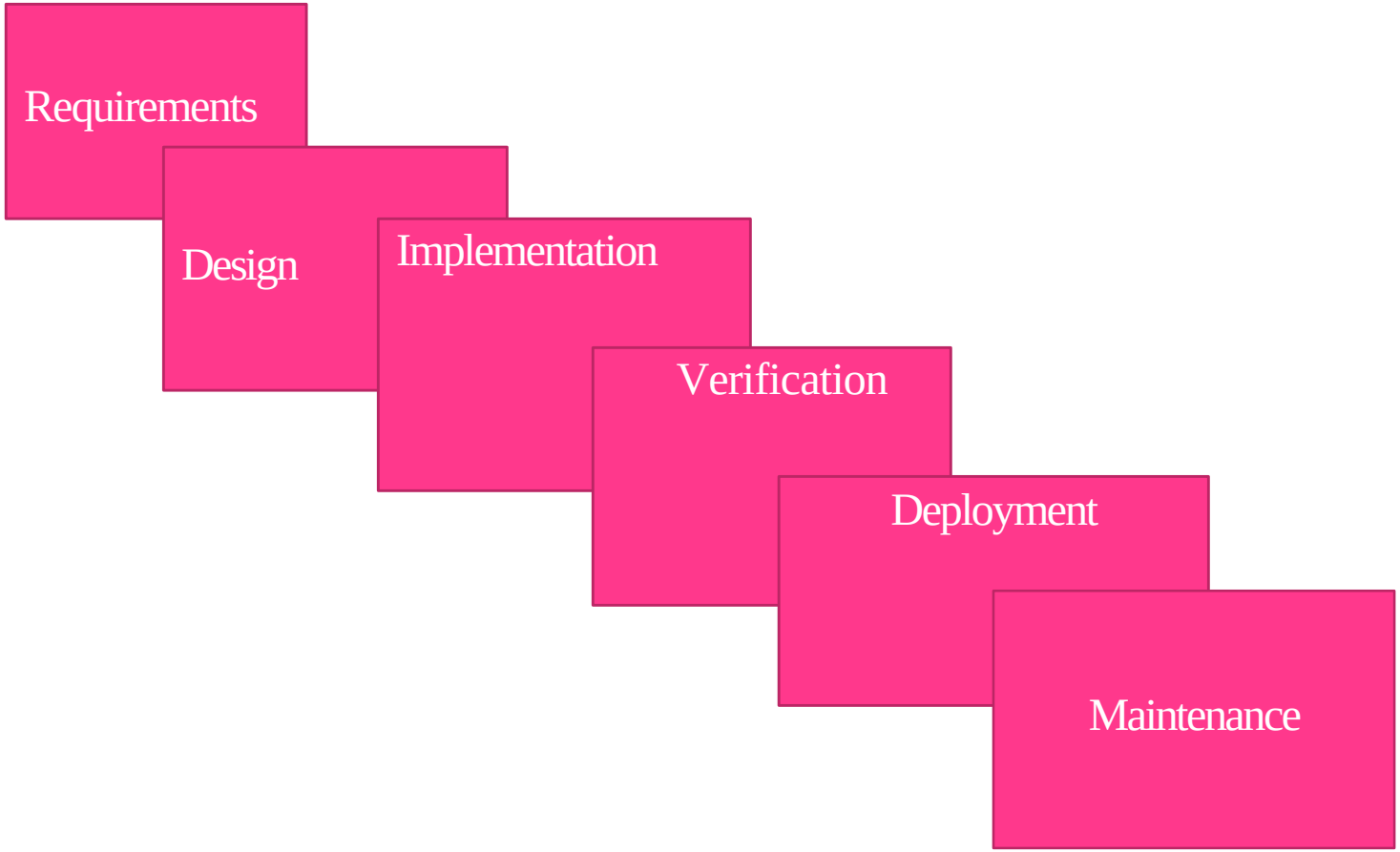
SASHIMI

- A modified version of the waterfall model.
- Also called sashimi waterfall or waterfall with overlapping phase.

- It is similar to the waterfall **except the steps are allowed to overlap.**

- Introducing feedback into the classical waterfall model.

SASHIMI


Requirements
Design
Implementation
Verification
Deployment
Maintenance

SASHIMI

- In a project's first phase, some requirements will be defined while you are still working on others.
- At that point, some of the team members can start designing the defined features while others continue working on the remaining requirements.

- The design for some parts of the application will be more or less finished but the design for other parts of the system won't be.
- At that point, some developers can start writing code for the designed parts of the system while others continue working on the rest of the design tasks, and may be or remaining requirements.

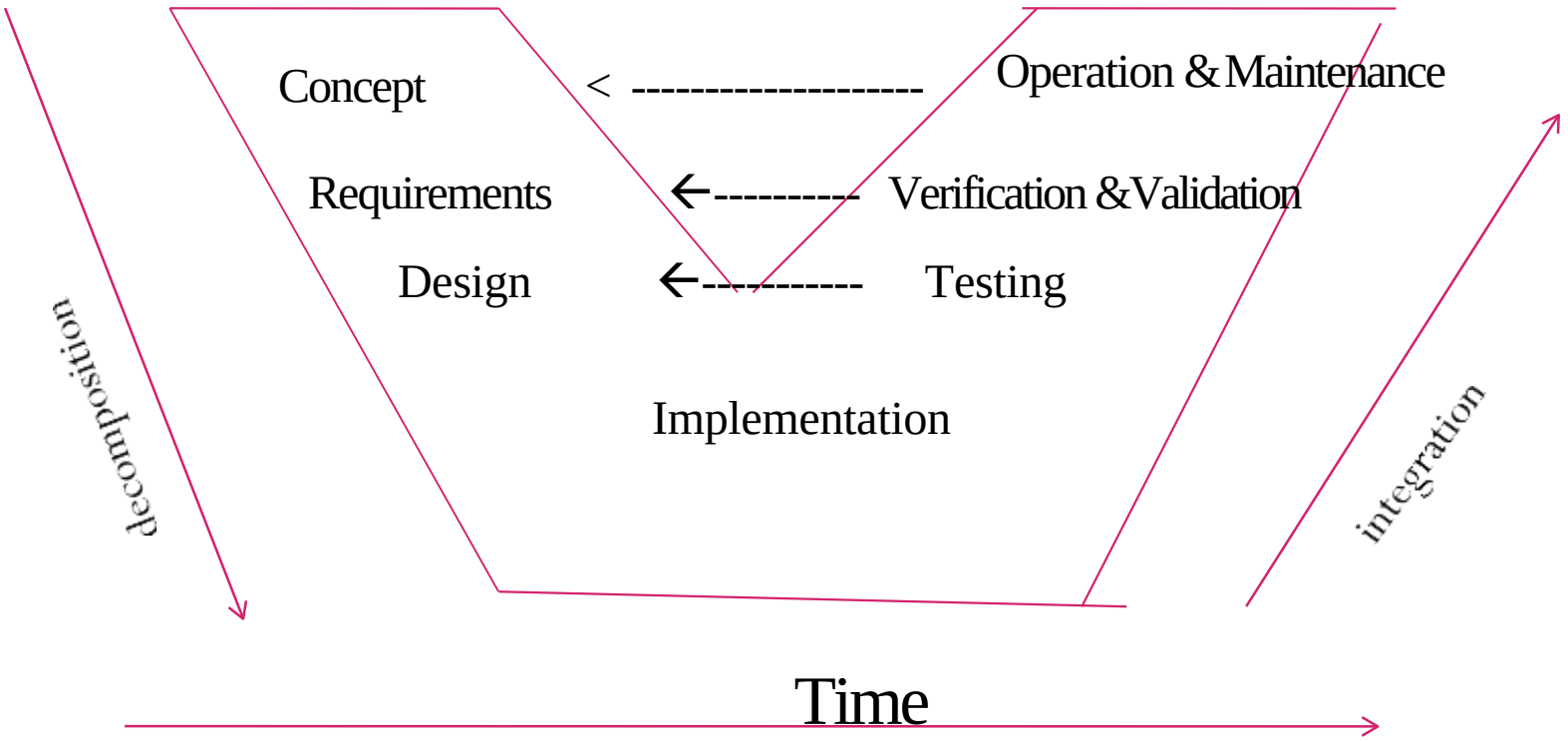
- There is a limitation for the overlap process.
- For example, we may want to delay deployment until the application is tested and verified.

SASHMI(advantages)

- People with different skills can focus on their specialities without waiting for others.
- If we discover during design that the requirements are impossible or need alterations, we can make the necessary changes.

V-MODEL

- Basically a waterfall model that's been bent into a **V shape**.



V-MODEL

- In this model verification and validation activities are carried out throughout the development life cycle, and therefore the chances bugs in the work products considerably reduce.

- The task on the left side of the V break the application down from its highest conceptual level into more and more detailed tasks.

- The process of breaking the application down into pieces is called **Decomposition**.

V-MODEL

- The tasks on the right side of the V consider the finished application at greater and greater levels of abstraction.

- At the lowest level, **Testing** verifies that the **code works**.
- At the next level, **verification** confirms that the application **satisfies the requirements**, and **validation** confirms that the application **meets customer needs**.

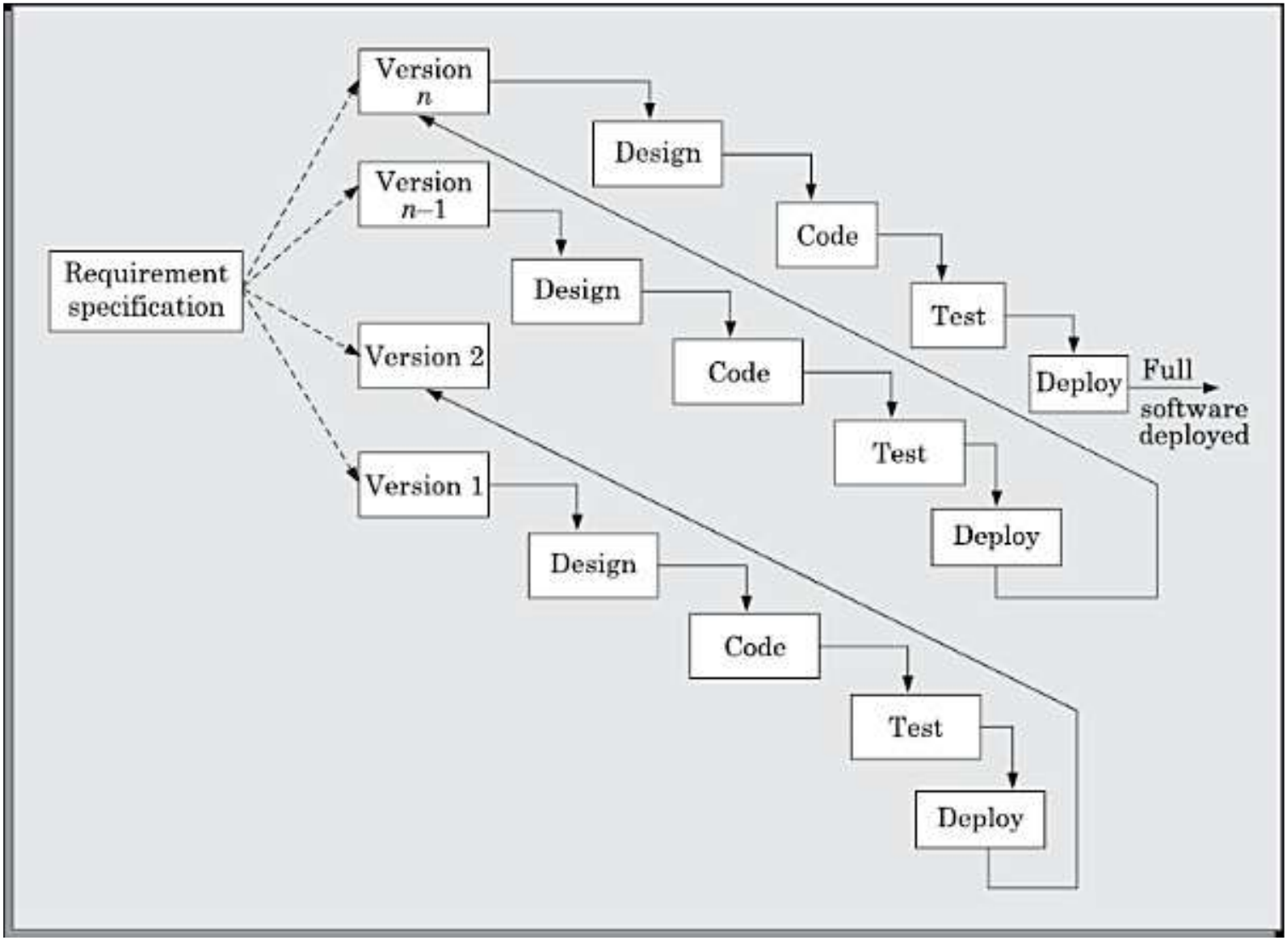
- The process of working back up to the conceptual top of the application is called **Integration**.

- Each of the tasks on the left corresponds to a task on the right with a similar level of abstraction.
- At the highest level, the initial concept corresponds to operation and maintenance.
- At the next level, the requirements correspond to verification and validation.
- Testing confirms that the design worked.

Incremental waterfall

- Also called multi-waterfall model.
- Uses a series of separate waterfall cascades.
- Each cascade ends with the delivery of a usable application called an increment.
- Each increment includes more features than the previous one.
- So, we are building the final application incrementally.

Incremental waterfall



- In the incremental life cycle model, the requirements of the software are first broken down into several modules or features that can be incrementally constructed and delivered.

- We can learn what did and didn't work well in the previous increment.

Contd…

- The incremental waterfall is **somewhat adaptive** model because its lets you to re-evaluate the direction at the start of each increment.
- The incremental waterfall model usually take long time to complete one iteration.

- We can change direction when we start a new increment, but within each increment the model runs predictively. In that sense, it is **not all that adaptive**.

Advantages

Error reduction:

- The core modules are used by the customer from the beginning and therefore these get tested thoroughly. This reduces chances of errors in the core modules of the final product, leading to greater reliability of the software.

Incremental resource deployment:

- This model obviates the need for the customer to commit large resources at one go for development of the system.
- It also saves the developing organisation from deploying large resources and manpower for a project in one go.

Prototyping Model

- It can be useful in iterative development.
- Prototype is a simplified model that demonstrates some behaviour of the project.
- Prototype does not work exactly the same way the finished application will work.
- However it lets the customer to see what the application will look like.
- After the customers experiment with the prototype, they can give us feedback to help refine the requirements.

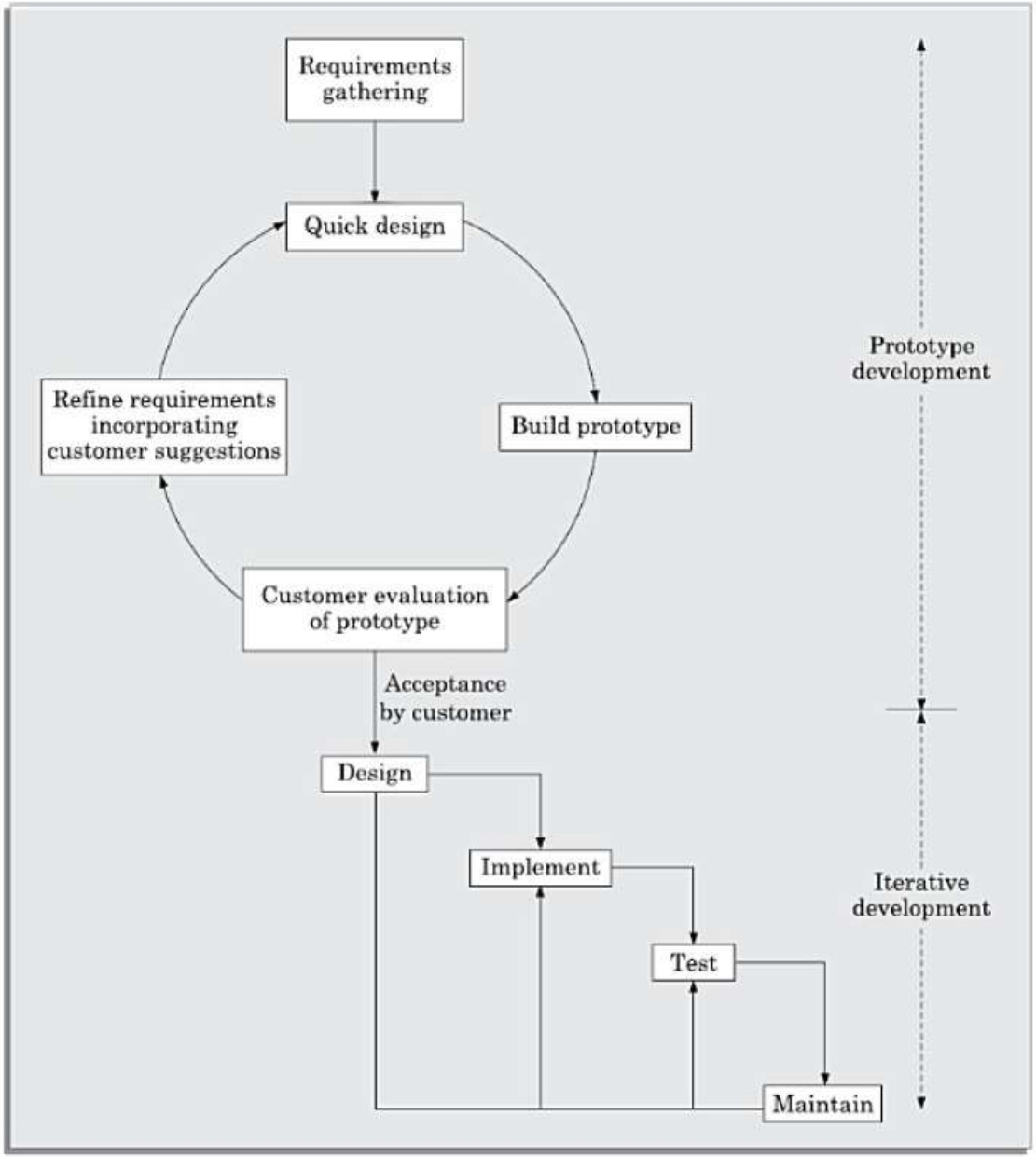


Figure 2.6: Prototyping model of software development.

Types of prototypes

Throwaway prototype:

In a throwaway prototype, we use the prototype to study some aspect of the system and then we throw it away and write code from scratch.

Evolutionary prototype:

In an evolutionary prototype, the prototype demonstrates some of the application's features.

As the project progresses, we refine those features and add new ones until the prototype morphs into the finished application.

- **Incremental prototype:**

In incremental prototyping, we build a collection of prototypes of that separately demonstrate the finished application's features.

We then combine the prototypes to build the finished application.

Advantages of Prototype

Improved requirements

- Prototypes allow customers to see what the finished application will look like. Often customers can spot problems andrequest changes earlier so the finished result is more useful to users.

- **Common vision**

  Prototypes let the customers and developers see the same preview of the finished application, so they are more likely to have a common vision of what the application should do and what it should look like.

- **Better design**

- Prototypes let the developers quickly explore specific pieces of the application to learn what they involve.

- It also help them to improve the design and make the final code more elegant and robust.

Disadvantages of Prototype

Narrowing vision

- People (customers and developers) tend to focus on a prototype's specific approach rather than on the problem it addresses.

Customer impatience

- A good prototype can make customers think that the finished application is just around the corner.

- **Scheduled pressure**
- If customers see a prototype that they think is mostly done, they may not understand that we need another year to finish and may pressure to shorten the schedule.

- **Raised expectation**
- Sometimes, a prototype may demonstrate features that won't be included in the application.
- For example, those features might turn out to be too hard. Sometimes, features are included to assess their value to users, and the features are dropped if they don't have enough benefit.

- **Attachment to code**
- Sometimes, developers become attached to the prototype's code.
- Initial code might have low quality.