

Software Life Cycle

A thick, horizontal yellow brushstroke underline that spans the width of the slide, positioned directly beneath the title text.

Software Life Cycle

- Software life cycle (or software process):
 - series of identifiable stages that a software product undergoes during its life time:
 - * Feasibility study
 - * requirements analysis and specification,
 - * design,
 - * coding,
 - * testing
 - * maintenance.

Life Cycle Model

- A software life cycle model (or process model):
 - a descriptive and diagrammatic model of software life cycle:
 - identifies all the activities required for product development,
 - establishes a precedence ordering among the different activities,
 - Divides life cycle into phases.

Why Model Life Cycle ?

- A written description:
 - forms a common understanding of activities among the software developers.
 - helps in identifying inconsistencies, redundancies, and omissions in the development process.
 - Helps in tailoring a process model for specific projects.

Life Cycle Model (CONT.)

- A life cycle model:
 - defines entry and exit criteria for every phase.
 - A phase is considered to be complete:
 - * only when all its exit criteria are satisfied.
- The phase exit criteria for the software requirements specification phase:
 - Software Requirements Specification (SRS) document is complete, reviewed, and approved by the customer.
- A phase can start:
 - only if its phase-entry criteria have been satisfied.

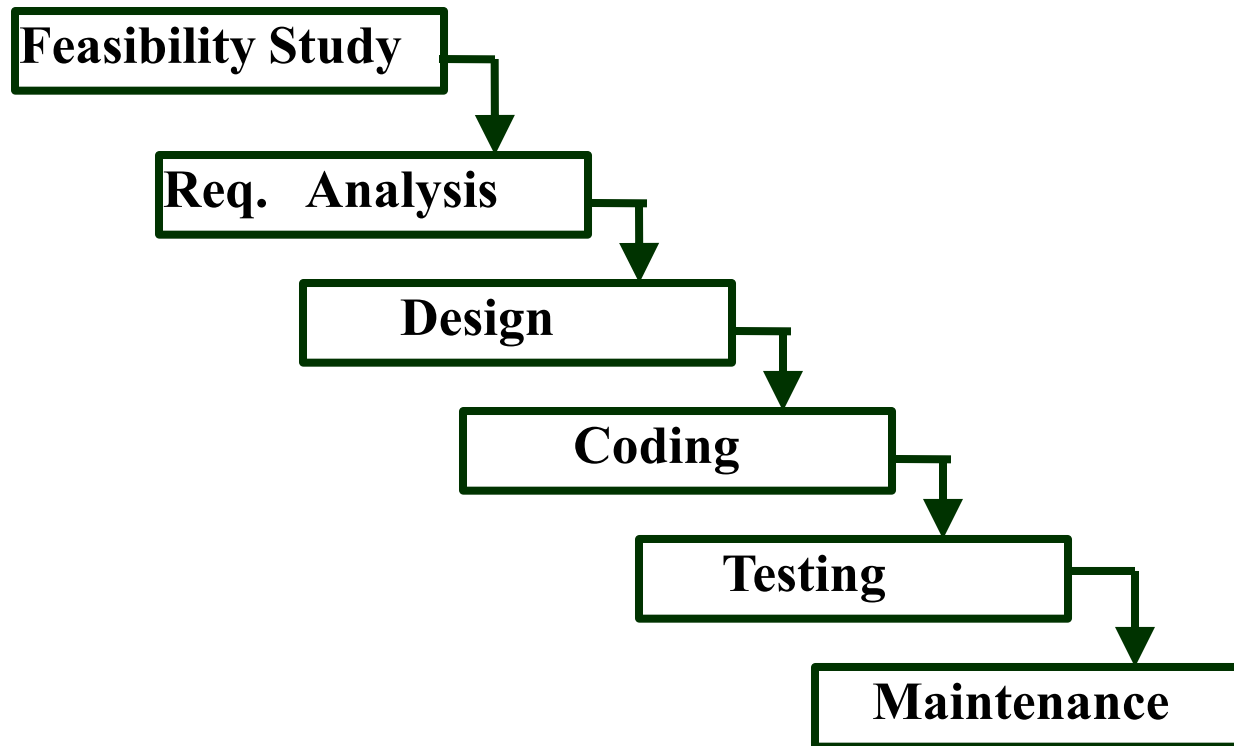
Different software life cycle models

- Many life cycle models have been proposed so far.
- Each of them has some advantages as well as some disadvantages.
 - Classical Waterfall Model
 - Iterative Waterfall Model
 - Prototyping Model
 - Evolutionary Model
 - Spiral Model

Different software life cycle models

- Many life cycle models have been proposed so far.
- Each of them has some advantages as well as some disadvantages.
 - Classical Waterfall Model
 - Iterative Waterfall Model
 - Prototyping Model
 - Evolutionary Model
 - Spiral Model

Classical Waterfall Model

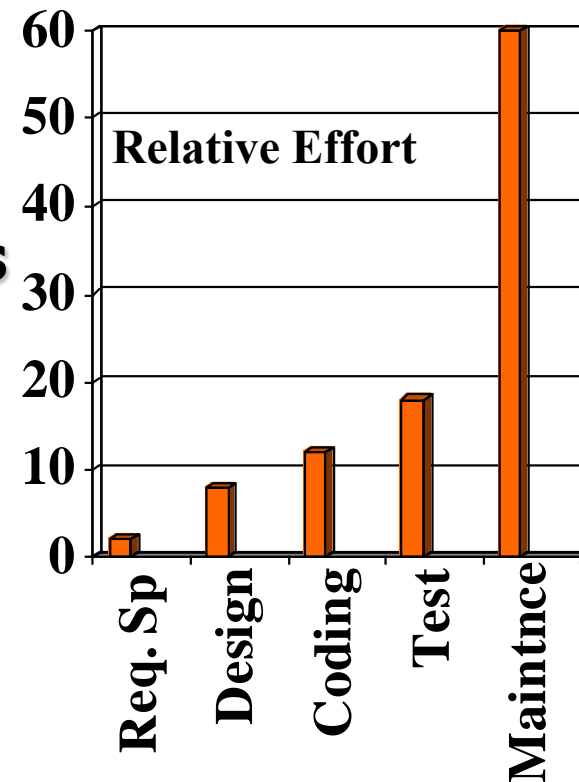


Classical Waterfall Model

- The Waterfall Model was the first Process Model to be introduced.
- It is also referred to as a **linear-sequential life cycle model**.
- It is very simple to understand and use.
- In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.
- All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases.

Relative Effort for Phases

- Phases between feasibility study and testing
 - known as **development phases**.
- Among all life cycle phases
 - **maintenance phase consumes maximum effort.**
- Among development phases,
 - testing phase consumes the maximum effort.



Classical Waterfall Model

(CONT.)

- **Most organizations usually define:**
 - standards on the outputs (deliverables) produced at the end of every phase
 - entry and exit criteria for every phase.
- **They also prescribe specific methodologies for:**
 - specification,
 - design,
 - testing,
 - project management, etc.

Feasibility Study

- **Main aim of feasibility study: determine whether developing the product**
 - financially worthwhile
 - technically feasible.
- **First roughly understand what the customer wants by:**
 - study different input data to the system and output data to be produced by the system
 - After an overall understanding of the problem they investigate the different solutions that are possible.
 - Examine each of the solutions in terms of resources required, cost of development and development time for each solution.

Activities during Feasibility Study

- **Perform a cost/benefit analysis:**
 - Based on this analysis pick the best solution and determine whether the solution is feasible financially and technically.
 - Check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

Requirements Analysis and Specification

- **Aim of this phase:**
 - understand the exact requirements of the customer,
 - document them properly.
- **Consists of two distinct activities:**
 - requirements gathering and analysis
 - requirements specification.

Requirements Gathering

- **Gathering relevant data:**
 - usually collected from the end-users through interviews and discussions.
 - For example, for a business accounting software:
 - * interview all the accountants of the organization to find out their requirements.

Requirements Analysis (CONT.)

- **Ambiguities and contradictions:**
 - must be identified
 - resolved by discussions with the customers.
- **Next, requirements are organized:**
 - into a **Software Requirements Specification (SRS)** document.
- **Engineers doing requirements analysis and specification are designated as analysts.**

Design

- **Design phase transforms requirements specification:**
 - into a form suitable for implementation in some programming language.
 - **during design phase, software architecture is derived from the SRS document.**
- **Two design approaches:**
 - traditional approach,
 - object oriented approach.

Traditional approach – Design Phase

- Consists of two activities:
 - **Structured analysis**
 - **Structured design**
- **Structured Analysis Activity**
 - Identify all the **functions** to be performed.
 - Identify **data flow** among the **functions**.
 - Decompose each function recursively into **sub-functions**.
 - * Identify data flow among the subfunctions as well.
- Carried out using **Data flow diagrams (DFDs)**.

Structured Design

- After structured analysis, carry out **structured design**:
 - **architectural design** (or high-level design)
 - **detailed design** (or low-level design).
- High-level design:
 - decompose the system into *modules*,
 - represent invocation relationships among the modules.
- Detailed design:
 - different modules designed in greater detail:
 - * data structures and algorithms for each module are designed.

Object Oriented Design

- First identify various **objects** (real world entities) occurring in the problem:
 - identify the **relationships** among the objects.
 - For example, the objects in a pay-roll software may be:
 - * employees,
 - * managers,
 - * pay-roll register,
 - * Departments, etc.
- Object structure
 - further refined to obtain the detailed design.
- OOD has several advantages:
 - lower development effort,
 - lower development time,
 - better maintainability.

Implementation

- **Purpose of implementation phase (aka **coding and unit testing** phase):**
 - **translate software design into source code.**
- During the implementation phase:
 - each module of the design is coded,
 - each module is **unit** tested
 - * tested independently as a stand alone unit, and debugged,
 - each module is documented.
- **The purpose of unit testing:**
 - test if individual modules work correctly.
- **The end product of implementation phase:**
 - **a set of program modules that have been tested individually.**

Integration and System Testing

- Different modules are integrated in a planned manner:
 - **modules are almost never integrated in one shot.**
 - Normally integration is carried out through a number of steps.
- During each integration step,
 - the partially integrated system is tested.



System Testing

- After all the modules have been successfully integrated and tested:
 - **system testing is carried out.**
- Goal of system testing:
 - ensure that the developed system functions according to its requirements as specified in the SRS document.

- System testing usually consists of three different kinds of testing activities:
 - α – testing: It is the system testing performed by the development team.
 - β – testing: It is the system testing performed by a friendly set of customers.
 - acceptance testing: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

Maintenance

- **Maintenance of any software product:**
 - requires much more effort than the effort to develop the product itself.
 - **development effort to maintenance effort is typically 40:60.**
- Corrective maintenance:
 - Correct errors which were not discovered during the product development phases.
- Perfective maintenance:
 - Improve implementation of the system
 - enhance functionalities of the system.
- Adaptive maintenance:
 - Port software to a new environment,
 - * e.g. to a new computer or to a new operating system.

Waterfall Model - Application

- Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors.
- Some situations where the use of Waterfall model is most appropriate are –
 - Requirements are very well documented, clear and fixed.
 - Product definition is stable.
 - Technology is understood and is not dynamic.
 - There are no ambiguous requirements.
 - Ample resources with required expertise are available to support the product.
 - The project is short.
- **Customer Relationship Management (CRM) systems, Human Resource Management Systems (HRMS), Supply Chain Management Systems, Inventory Management Systems, Point of Sales (POS) systems for Retail chains etc**

Advantages of the Waterfall Model

- The advantages of waterfall development are that it allows for departmentalization and control.
 - Each phase has specific deliverables and a review process.
 - Phases are processed and completed one at a time.
 - Works well for smaller projects where requirements are very well understood.
 - Clearly defined stages.
 - Well understood milestones.
 - Process and results are well documented.

Waterfall Model - Disadvantages

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- Cannot accommodate changing requirements.
- Integration is done as a "big-bang" at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Iterative Waterfall Model

- **Classical waterfall model is idealistic:**
 - assumes that no defect is introduced during any development activity.
 - in practice:
 - * defects do get introduced in almost every phase of the life cycle.
- Defects usually get detected much later in the life cycle:
 - For example, a design defect might go unnoticed till the coding or testing phase.
- **Therefore we need feedback paths in the classical waterfall model.**

Iterative Waterfall Model

(CONT.)

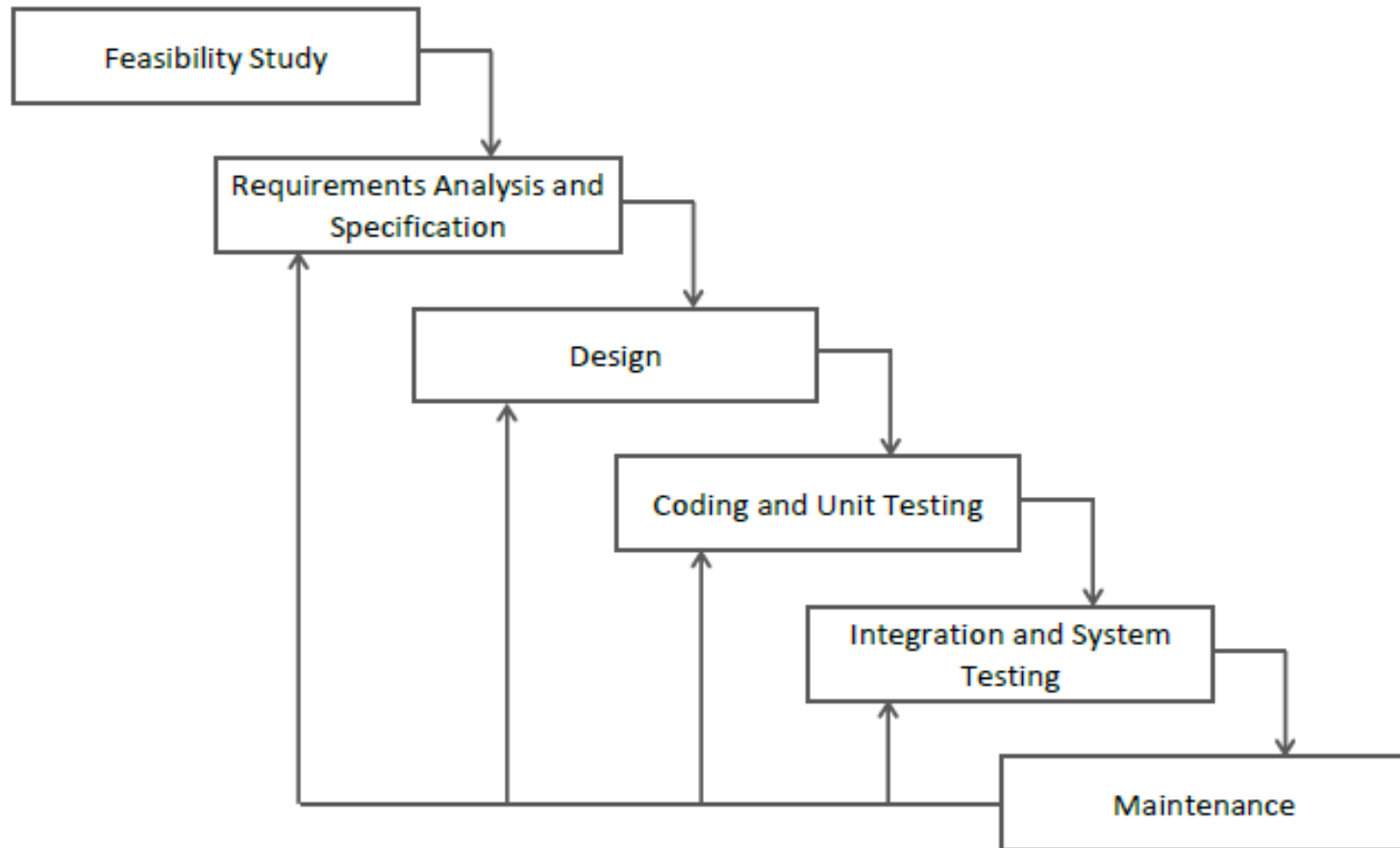


Figure 2: Iterative Waterfall Model

Iterative Waterfall Model

- **Errors should be detected**
 - in the same phase in which they are introduced.
- **For example:**
 - **if a design problem is detected in the design phase itself,**
 - the problem can be taken care of much more easily
 - than say if it is identified at the end of the integration and system testing phase.

Phase containment of errors

- Reason: rework must be carried out not only to the design but also to code and test phases.
- The principle of detecting errors as close to its point of introduction as possible:
 - is known as **phase containment of errors.**
- Iterative waterfall model is by far the most widely used model.
 - Almost every other model is derived from the waterfall model.

Prototyping Model

- **Before starting actual development,**
 - **a working prototype of the system should first be built.**
- **A prototype is a toy implementation of a system:**
 - **limited functional capabilities,**
 - **low reliability,**
 - **inefficient performance.**

Reasons for developing a prototype

- Illustrate to the customer:
 - input data formats, messages, reports, or interactive dialogs.
- Examine technical issues associated with product development:
 - Often major design decisions depend on issues like:
 - * response time of a hardware controller,
 - * efficiency of a sorting algorithm, etc.

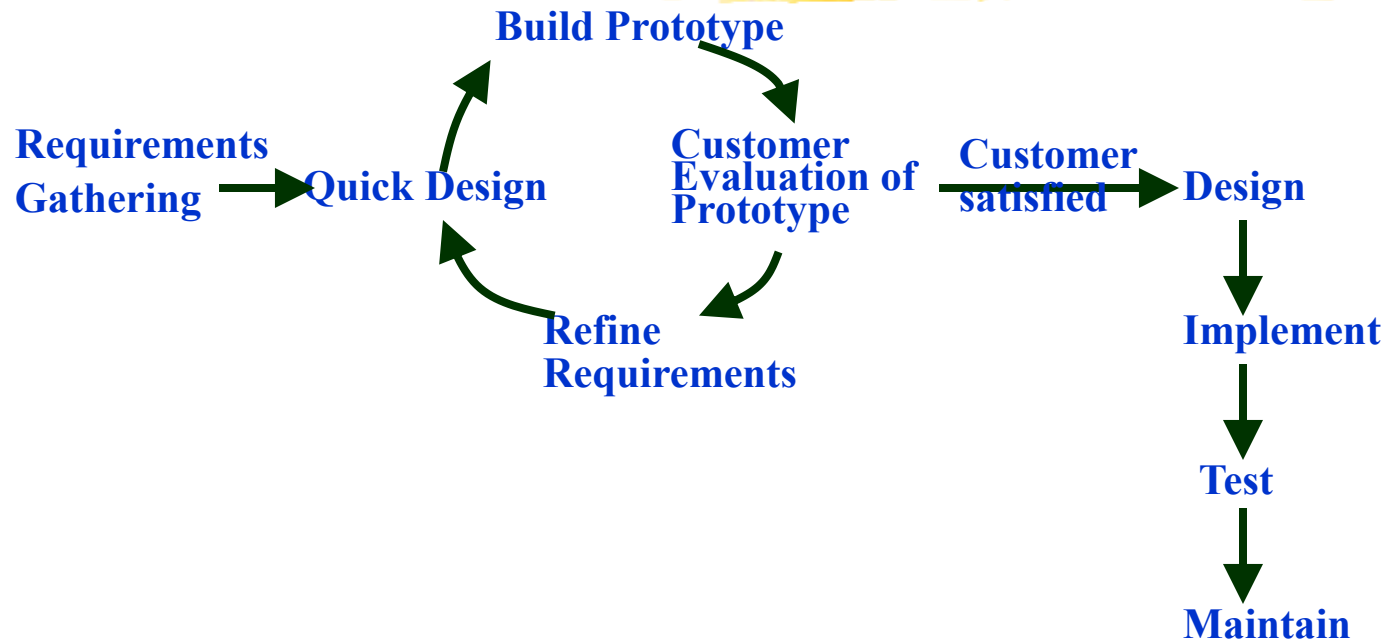
Prototyping Model (CONT.)

- **The third reason for developing a prototype is:**
 - **it is impossible to ``get it right'' the first time,**
 - **we must plan to throw away the first product**
 - * **if we want to develop a good product.**
- **Carry out a quick design.**
- **Prototype model is built using several short-cuts:**
 - Short-cuts might involve using inefficient, inaccurate, or dummy functions.
 - * A function may use a table look-up rather than performing the actual computations.

Prototyping Model (CONT.)

- **The developed prototype is submitted to the customer for his evaluation:**
 - Based on the user feedback, requirements are refined.
 - This cycle continues until the user approves the prototype.
- **The actual system is developed using the classical waterfall approach.**

Prototyping Model (CONT.)



Prototyping Model (CONT.)

- Requirements analysis and specification phase becomes redundant:
 - final working prototype (with all user feedbacks incorporated) serves as an **animated requirements specification**.
- Design and code for the prototype is usually thrown away:
 - However, the experience gathered from developing the prototype helps a great deal while developing the actual product.

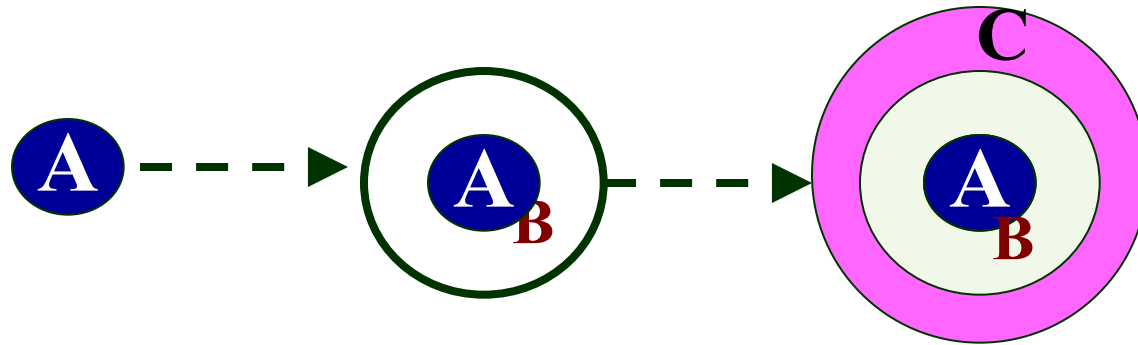
Prototyping Model (CONT.)

- **Even though construction of a working prototype model involves additional cost --- overall development cost might be lower for:**
 - systems with unclear user requirements,
 - systems with unresolved technical issues.
- **Many user requirements get properly defined and technical issues get resolved:**
 - these would have appeared later as change requests and resulted in incurring massive redesign costs.

Evolutionary Model

- **Evolutionary model (aka successive versions or incremental model):**
 - The system is broken down into several modules which can be incrementally implemented and delivered.
- **First develop the core modules of the system.**
- **The initial product skeleton is refined into increasing levels of capability:**
 - by adding new functionalities in successive versions.
- **Successive version of the product:**
 - **functioning systems capable of performing some useful work.**
 - **A new release may include new functionality:**
 - * also existing functionality in the current release might have been enhanced.

Evolutionary Model (CONT.)



Advantages of Evolutionary Model

- **Users get a chance to experiment with a partially developed system:**
 - much before the full working version is released,
- **Helps finding exact user requirements:**
 - much before fully working system is developed.
- **Core modules get tested thoroughly:**
 - reduces chances of errors in final product.

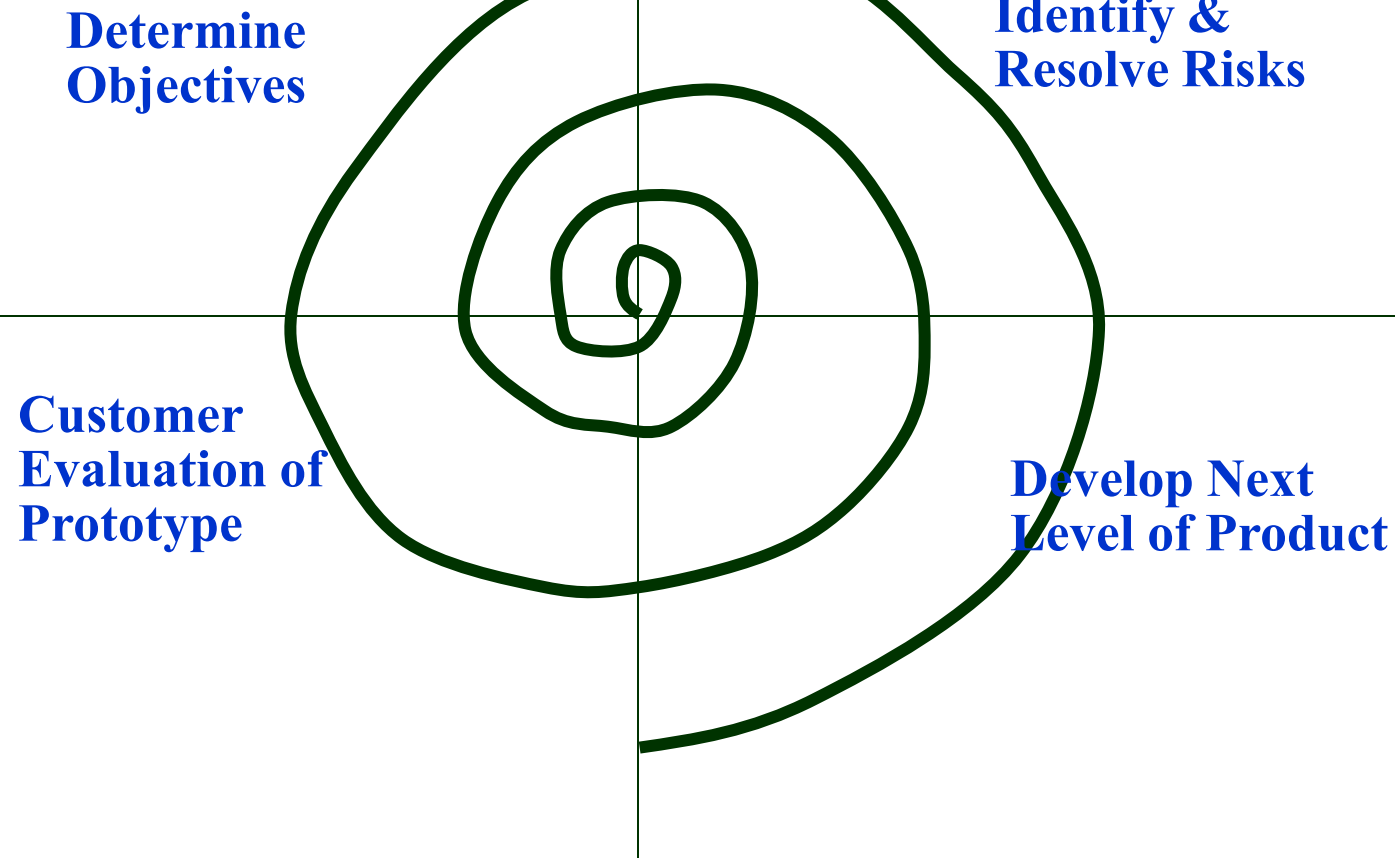
Disadvantages of Evolutionary Model

- Often, difficult to subdivide problems into functional units:
 - which can be incrementally implemented and delivered.
 - evolutionary model is useful for very large problems,
 - * where it is easier to find modules for incremental implementation.

Evolutionary Model with Iteration

- **Many organizations use a combination of iterative and incremental development:**
 - a new release may include new functionality
 - existing functionality from the current release may also have been modified.
- **Several advantages:**
 - Training can start on an earlier release
 - * customer feedback taken into account
 - Markets can be created:
 - * for functionality that has never been offered.
 - Frequent releases allow developers to fix unanticipated problems quickly.

Spiral Model (CONT.)



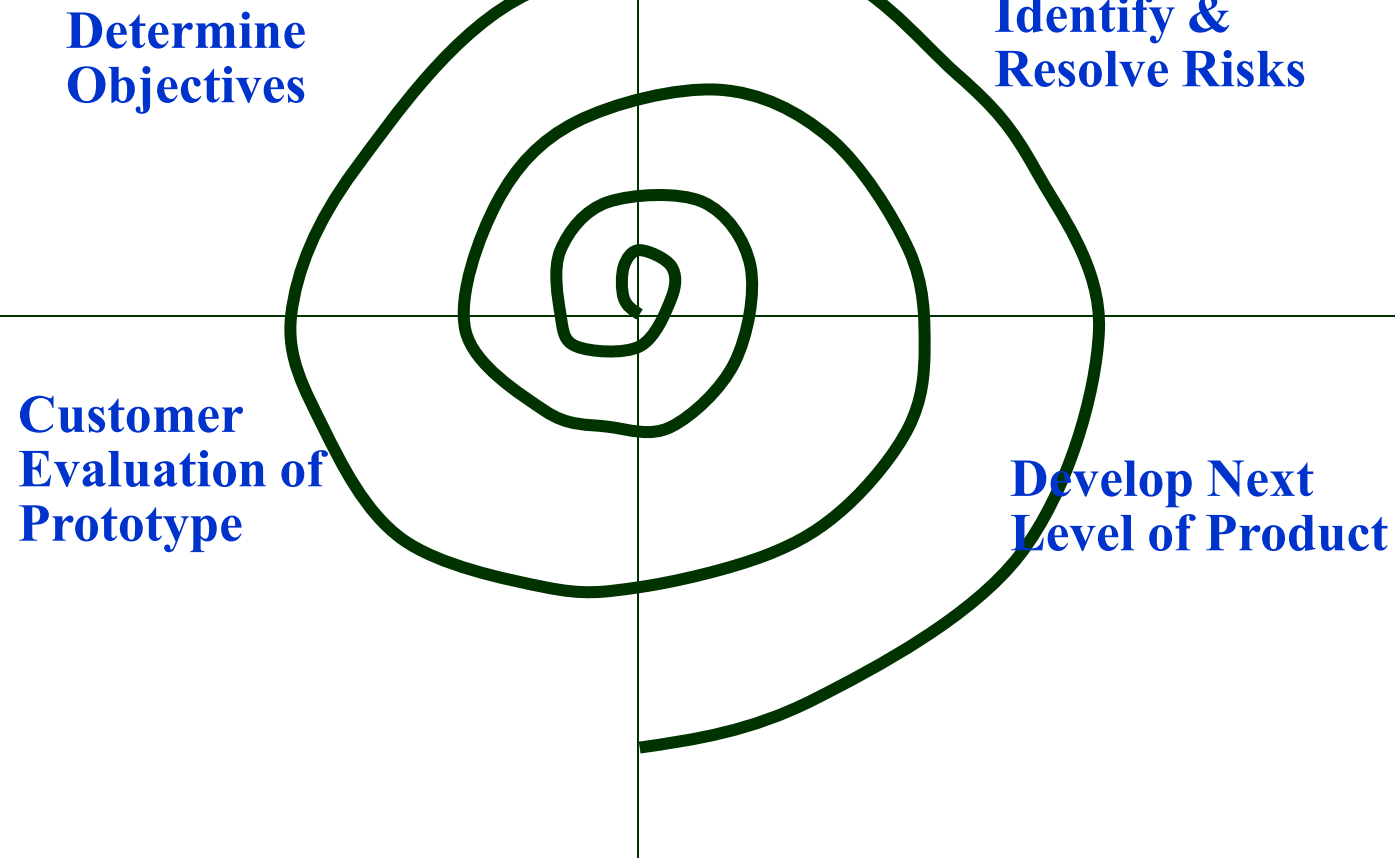
Spiral Model

- **Each loop of the spiral represents a phase of the software process:**
 - the innermost loop might be concerned with system feasibility,
 - the next loop with system requirements definition,
 - the next one with system design, and so on.
- **The exact number of loops in the spiral is not fixed.**

Spiral Model (CONT.)

- **The team must decide:**
 - **how to structure the project into phases.**
- **Start work using some generic model:**
 - **add extra phases**
 - * **for specific projects or when problems are identified during a project.**
- **Each loop in the spiral is split into four sectors (quadrants).**

Spiral Model (CONT.)



Objective Setting (First Quadrant)

- Identify objectives of the phase,
- Examine the **risks** associated with these objectives.
 - Risk:
 - * any adverse circumstance that might hamper successful completion of a software project.
- Find alternate solutions possible.

Risk Assessment and Reduction (Second Quadrant)

- **For each identified project risk,**
 - **a detailed analysis is carried out.**
- **Steps are taken to reduce the risk.**
- **For example, if there is a risk that the requirements are inappropriate:**
 - **a prototype system may be developed.**

Spiral Model (CONT.)

- **Development and Validation (Third quadrant):**
 - develop and validate the next level of the product.
- **Review and Planning (Fourth quadrant):**
 - review the results achieved so far with the customer and plan the next iteration around the spiral.
- **With each iteration around the spiral:**
 - progressively more complete version of the software gets built.

Spiral Model as a meta model

- **Subsumes all discussed models:**
 - a single loop spiral represents waterfall model.
 - uses an evolutionary approach --
 - * iterations through the spiral are evolutionary levels.
 - enables understanding and reacting to risks during each iteration along the spiral.
 - uses:
 - * prototyping as a risk reduction mechanism
 - * retains the step-wise approach of the waterfall model.

Advantages of Spiral Model

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages

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

Circumstances to use spiral model

- The spiral model is called a meta model since it encompasses all other life cycle models.
- Risk handling is inherently built into this model.
- The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks.
- However, this model is much more complex than the other models – **this is probably a factor deterring its use in ordinary projects.**

Comparison of Different Life Cycle Models

- **Iterative waterfall model**
 - most widely used model.
 - But, suitable only for well-understood problems.
- **Prototype model is suitable for projects not well understood:**
 - **user requirements**
 - **technical aspects**

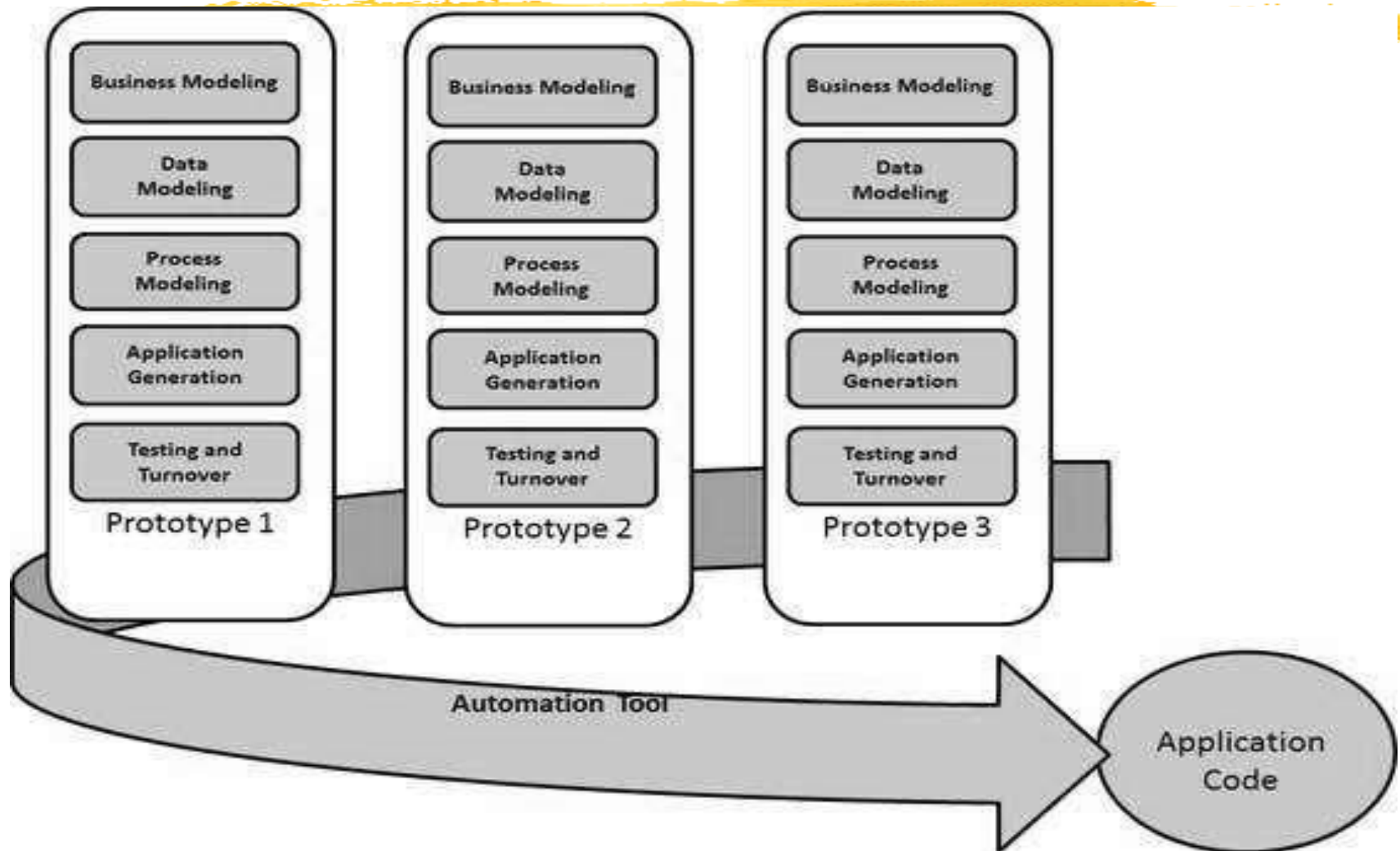
Comparison of Different Life Cycle Models (CONT.)

- **Evolutionary model is suitable for large problems:**
 - can be decomposed into a set of modules that can be incrementally implemented,
 - incremental delivery of the system is acceptable to the customer.
- **The spiral model:**
 - suitable for development of technically challenging software products that are subject to several kinds of risks.

RAD

- RAD model is Rapid Application Development model.
- Based on prototyping and iterative development with no specific planning involved.
- In RAD model the components or functions are developed in parallel as if they were mini projects.
- Focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.

The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.



- The phases in the rapid application development (RAD) model are:
 - **Business Modeling:** Model should be designed based on the information available from different business activities.
 - **Data Modeling:** All the required and necessary data based on business analysis are identified in data modeling phase.
 - **Process Modeling:** In this phase all the data modification process is defined. Process descriptions for adding, deleting, retrieving or modifying a data object are given.
 - **Application Modeling:** The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.
 - **Testing and turnover:** All the testing activates are performed to test the developed application.

- **Advantages of RAD Model:**

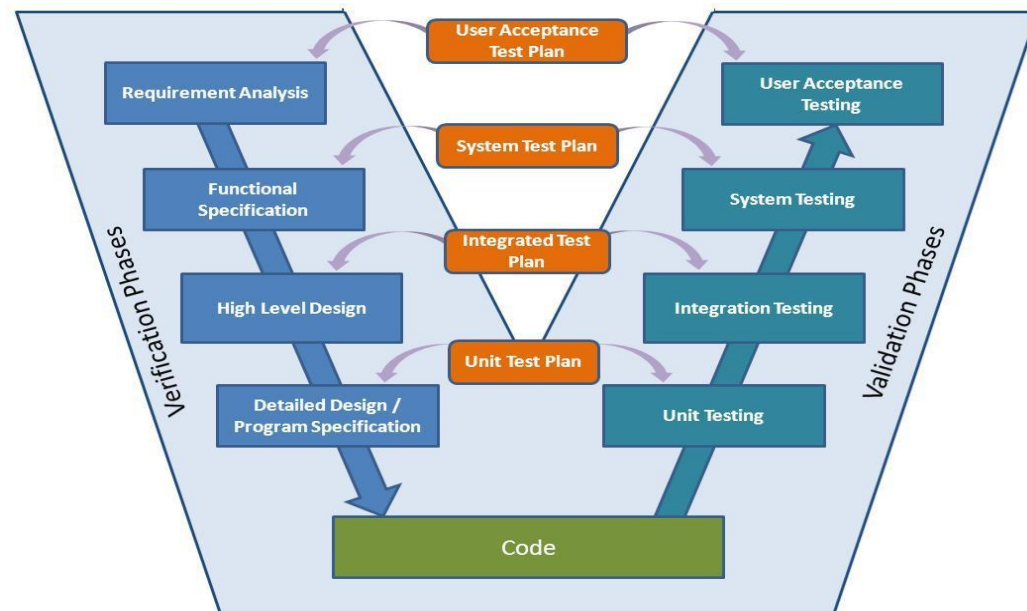
- Fast application development and delivery.
- Least testing activity required.
- Visualization of progress.
- Review by the client from the very beginning of development so very less chance to miss the requirements.

- **Disadvantages of RAD Model:**

- High skilled resources required.
- On each development phase client's feedback required.
- Automated code generation is very costly.
- Difficult to manage.

V-Shaped Model

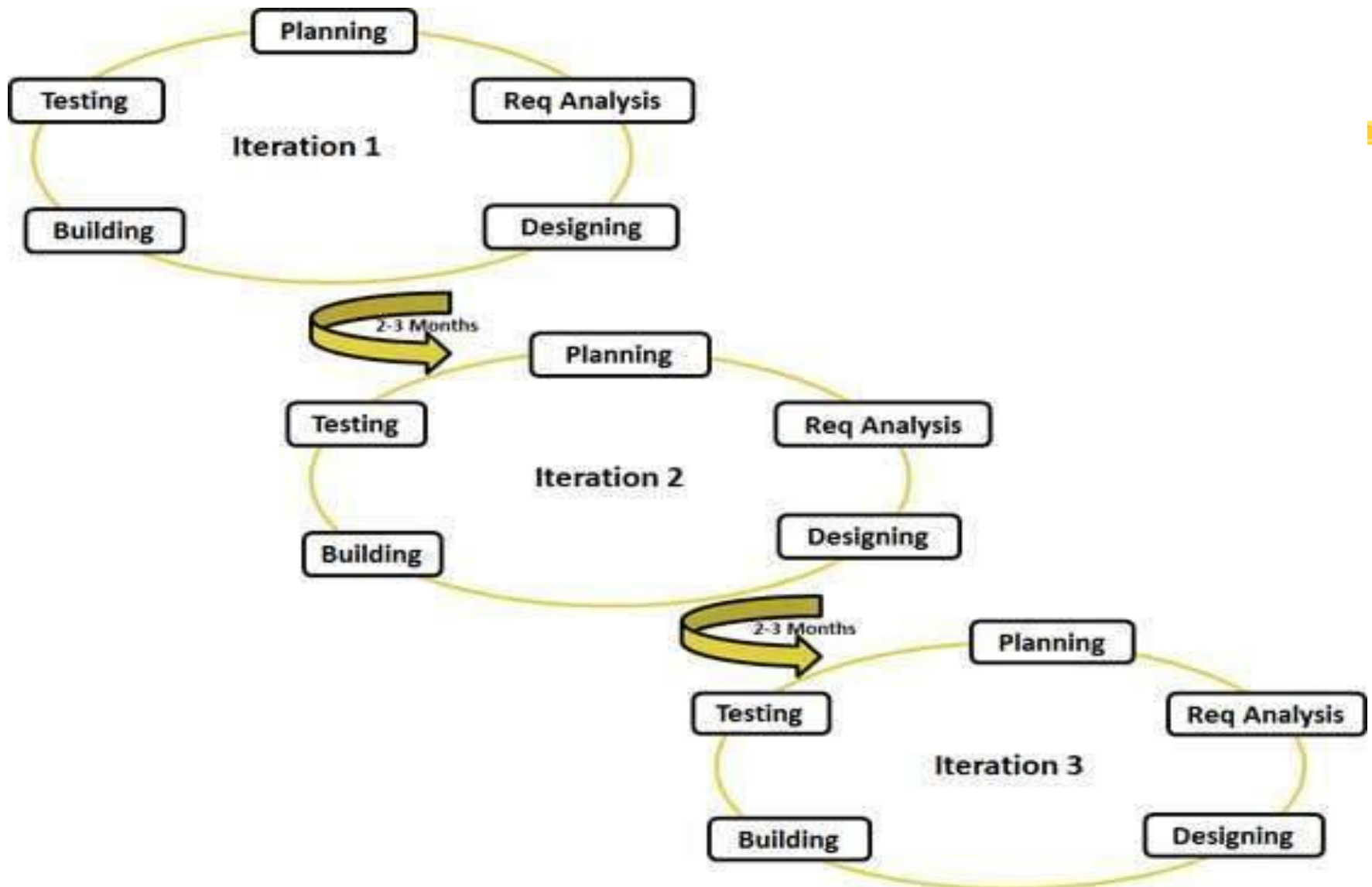
- The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as **Verification and Validation model**.
- The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development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



- The usage
 - Software requirements clearly defined and known
 - Software development technologies and tools is well-known
- **Advantages**
 - Simple and easy to use.
 - Each phase has specific deliverables.
 - Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
 - Verification and validation of the product in early stages of product development

Agile Model

- Combination of **iterative and incremental process models** with focus on process adaptability and customer satisfaction by rapid delivery of working software product.
- Agile Methods break the product into **small incremental builds**.
- These builds are provided in iterations. Each iteration typically lasts from about one to three weeks.
- Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.
- At the end of the iteration a working product is displayed to the customer and important stakeholders.



Advantages

- Is a very realistic approach to software development
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.

Disadvantages

- Not suitable for handling complex dependencies.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

Software Development Methodology: Agile

Merits and limitations of Agile approach (MyPMExpert, 2009)

Merits	Limitations
<ul style="list-style-type: none">• Agile methodology is adaptive and hence it can adapt well with changing requirements.• Dedicated time and effort is not required because customer requirement may change• Customer can give continuous inputs and have face to face interactions with the team.• Agile method does not give room for guesswork, documentation is exhaustive but crisp• Customer satisfaction is ensured	<ul style="list-style-type: none">• Difficult to scale when projects are large where documentation is needed• Agile teams need to have experience and skills• In some deliverables it is problematic to evaluate as required at beginning of SDLC• Design is not much emphasized• The project can be easily distracted when the customer is not clear on outcomes• Testing and test construction is difficult and needs specialized skills

Software Life Cycle


- A software life cycle model (also called process model) is a **descriptive and diagrammatic representation** of the software life cycle.
- Represents all the **activities** required to make a software product transit through its life cycle phases.
- Captures the **order in which these activities** are to be undertaken.
- In other words maps the different activities performed on a software product from its inception to retirement.

- Different life cycle models may map the basic development activities to phases in different ways.
- Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models.
- During any life cycle phase, more than one activity may also be carried out.
- For example, the **design phase** might consist of the **structured analysis** activity followed by the **structured design activity**.

Need for a software life cycle model

- Development team must identify a suitable life cycle model for the particular project and then adhere to it.
- Without using of a particular life cycle model the development of a software product would not be in a systematic and disciplined manner.
- When a software product is being developed by a team there must be a clear understanding among team members about when and what to do.
- Otherwise it would lead to chaos and project failure.

- Suppose a software development problem is divided into several parts and assigned to the team members.
- From then on, suppose the team members are allowed the freedom to develop the parts assigned to them in whatever way they like.
- It is possible that one member might start writing the code for his part,
- Another might decide to prepare the test documents first,
- And some other engineer might begin with the design phase of the parts assigned to him.
- **This would be one of the perfect recipes for project failure.**

- A software life cycle model defines **entry and exit criteria for every phase.**
- 
- A phase can start only if its **phase-entry criteria** have been satisfied.
 - So without software life cycle model the **entry and exit criteria for a phase cannot be recognized.**
 - Without software life cycle models it becomes difficult for software project managers to monitor the progress of the project.