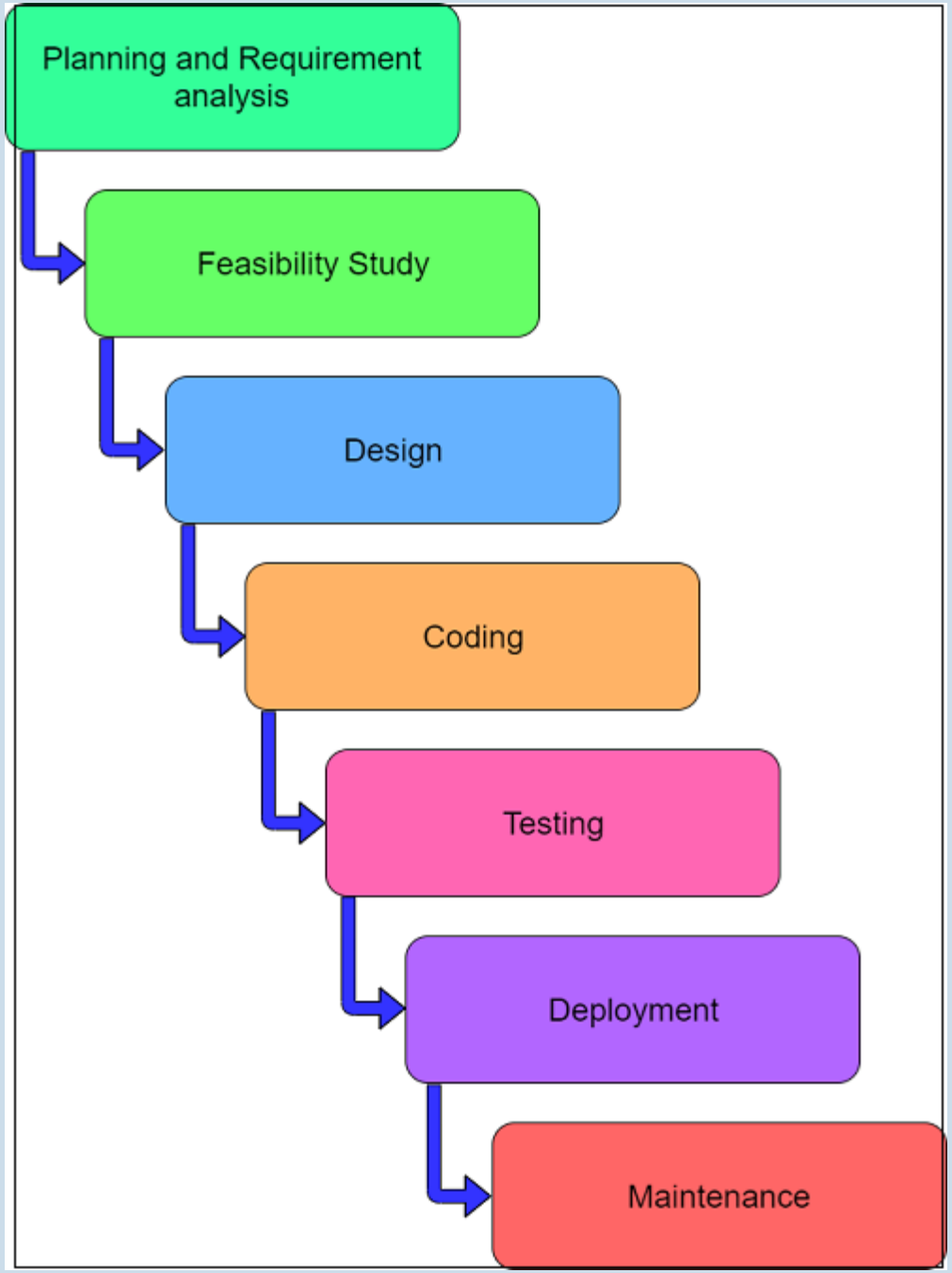
SDLC

SDLC PHASES
THE ENTIRE SDLC PROCESS DIVIDED INTO THE FOLLOWING STAGES:

- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study:
- Phase 3: Design:
- Phase 4: Coding:
- Phase 5: Testing:
- Phase 6: Installation/Deployment:
- Phase 7: Maintenance:


Planning and Requirement analysis
Feasibility Study
Design
Coding
Testing
Deployment
Maintenance

PHASE 1: REQUIREMENT COLLECTION AND ANALYSIS:

- The requirement collection is the first stage in the SDLC process.
- It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry.
- Planning for the quality assurance requirements and recognition of the risks involved is also done at this stage.
- This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.
- Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

PHASE 2: FEASIBILITY STUDY

Once the requirement analysis phase is completed the next SDLC step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of feasibilities checks:

- Economic: Can we complete the project within the budget or not?
- Legal: Can we handle this project as cyber law and other regulatory framework/compliances.
- Operation feasibility: Can we create operations which is expected by the client?
- Technical: Need to check whether the current computer system can support the software
- Schedule: Decide that the project can be completed within the given schedule or not.

PHASE 3: DESIGN

- In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.
- This design phase serves as input for the next phase of the model.
- There are two kinds of design documents developed in this phase:
  - High-Level Design (HLD)
  - Low-Level Design(LLD)

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design(LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

PHASE 4: CODING

- Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language.
- In the coding phase, tasks are divided into units or modules and assigned to the various developers.
- It is the longest phase of the Software Development Life Cycle process.
- In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

PHASE 5: TESTING

- Once the software is complete, and it is deployed in the testing environment.
- The testing team starts testing the functionality of the entire system.
- This is done to verify that the entire application works according to the customer requirement.
- During this phase, QA and testing team may find some bugs/defects which they communicate to developers.
- The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

PHASE 6: INSTALLATION/DEPLOYMENT

- Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts.
- Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

PHASE 7: MAINTENANCE

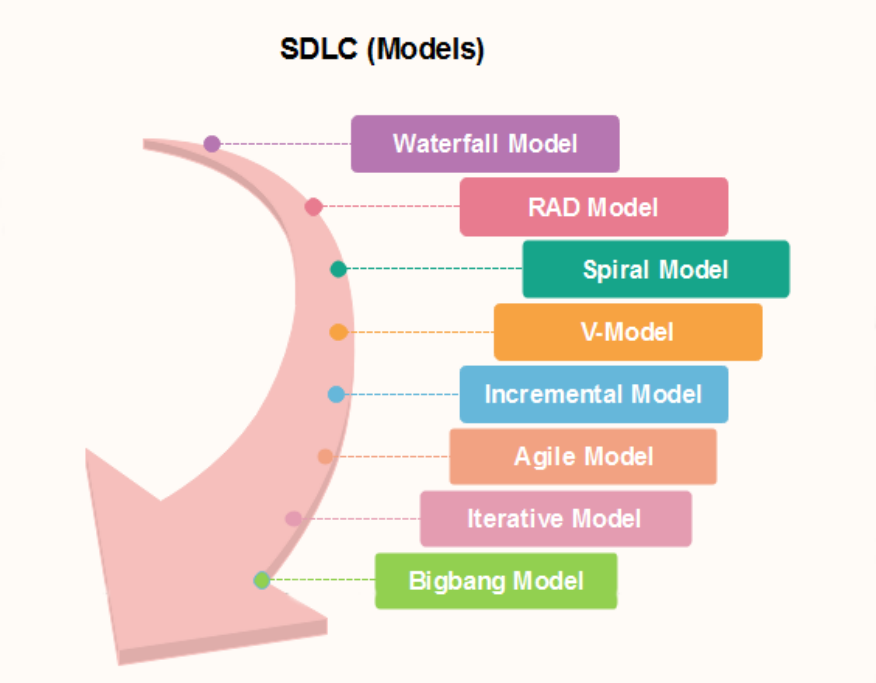
Once the system is deployed, and customers start using the developed system, following 3 activities occur

- Bug fixing - bugs are reported because of some scenarios which are not tested at all
- Upgrade - Upgrading the application to the newer versions of the Software
- Enhancement - Adding some new features into the existing software

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

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. A few important and commonly used life cycle models are as follows:



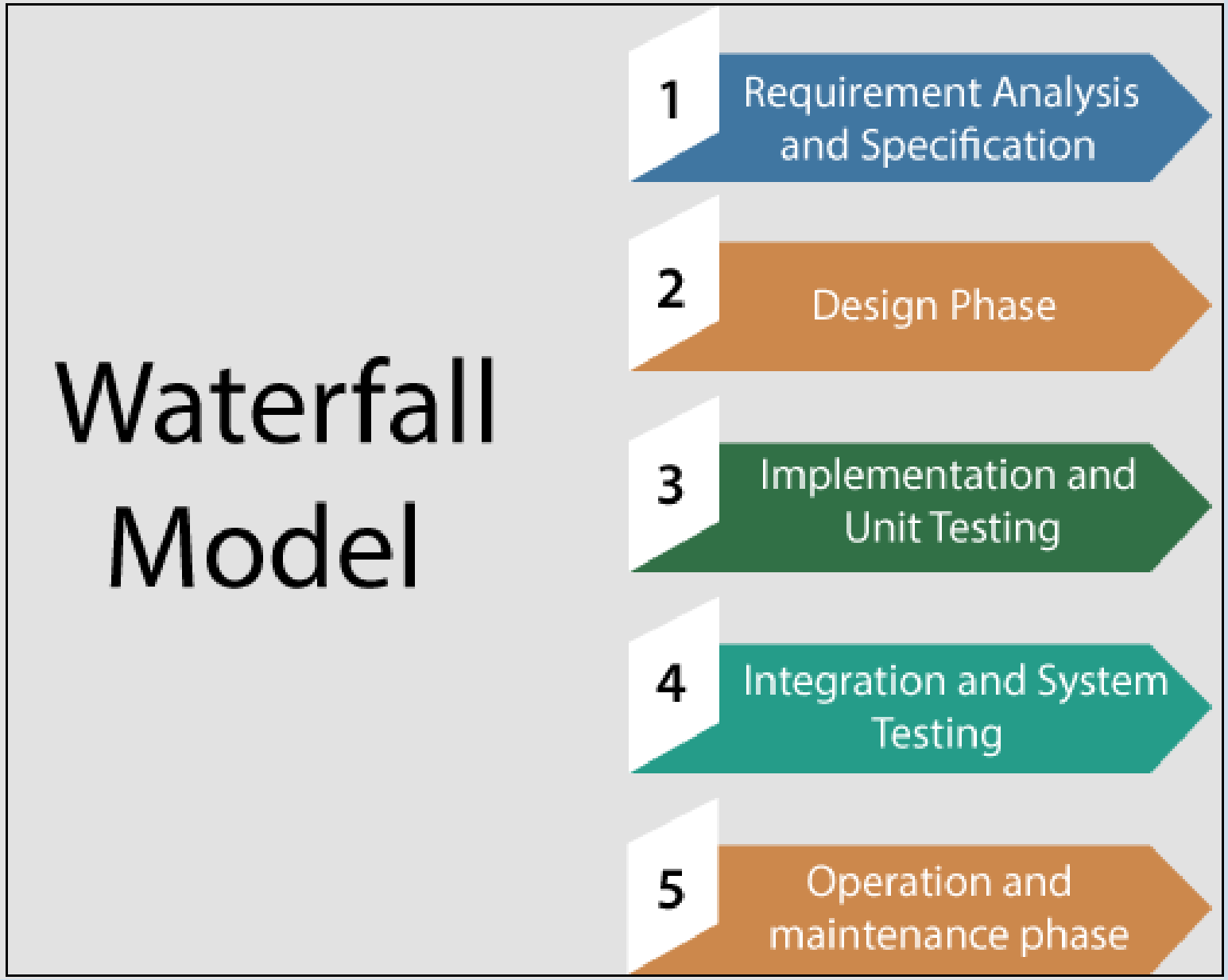
WATERFALL MODEL

- Winston Royce introduced the Waterfall Model in 1970.
- The most popular model is the waterfall model. This model has five phases:
  - Requirement analysis and specification
  - Design
  - Implementation & unit testing
  - Integration and system testing phase
  - Operation and maintenance

  These phases often occur one after the other. The developer should complete each step before the next phase starts.

  This model is called the "Waterfall Model" because its diagrammatic representation resembles a waterfall.

Waterfall Model



Requirement analysis and specification phase:-

- This phase aims to understand and properly document the exact requirements of the customer.
- This task is usually carried out in collaboration with the customer, as the aim is to record all features, functionality, and configuration specifications for the application.
- The requirements describe the “what” of the system, not the “how” of the system. This process produces a large file, written in a natural language describes what the program will do without explaining how it will be accomplished.
- The resultant document is known as a software requirement specification (SRS) document.
- The SRS document may act as a contract between the developer and the customer. If the developer fails to implement a full set of requirements, it may fail to implement the contracted set of requirements.

Design phase:-

- The SRS document is produced in the previous phase, which contains the exact requirements of the customer.
- The purpose of this step is to convert the requirement specification into a structure that is suitable for implementation in some programming languages.
- Hence, overall software architecture is defined, and the high level and detailed design work are performed.
- This work is documented and known as a software design description (SDD) document. The information included in the Software Design Description should be sufficient to start the coding phase.

Implementation and unit testing phase:-

- During this phase, the design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because of all the information needed by the software developers contained in the SDD.
- During testing, the primary activities are centered around the examination and modification of the code. Small modules were initially tested in isolation from the rest of the software product.
- There are problems associated with the evaluation of unit tests in isolation. How do we run a module without anything to label it, to call it, or, possibly, to intermediate output values obtained during execution?
- These problems are resolved at this point, and the modules are checked after writing some overhead code.

Integration and system testing:-

- This is a very important phase. Good testing can help to deliver higher quality software products, more happy customers, lower maintenance costs, and more accurate and reliable performance.
- It is a very expensive activity and consumes one-third to at least one half the value of a typical development project.
- As we know, the purpose of unit testing is to determine whether each independent module is implemented correctly.
- It gives very little chance to determine if the interface between modules is also correct and, therefore, an integration testing is performed. System testing involves the testing of the entire system, whereas software is a part of the system.
- This is essential to build confidence in the developers before the software is delivered to the customer or released in the market.

Operation and maintenance phase:-

- Software maintenance is a challenge that every team has to face when the software is deployed and installed to the customer's site.
- Therefore, the release of software inaugurates the operation and maintenance phase of the life cycle.
- The time and effort required to maintain the operating program after launch are very important.
- Despite the fact it is a very challenging task, it is routinely the poorly managed headache that nobody wants to face.
- Software maintenance is a very broad activity that includes error correction, enhancement of capabilities, and optimizations.
- The purpose of this phase is to maintain the value of the software over time. This stage can last from 5 to 50 years, while development can last from 1 to 3 years.

It is easy to reinforce the model "define before design" and "design before code" design. This model sometimes expects complete and accurate requirements, which is unrealistic. There is also no risk assessment involved.

When to use SDLC Waterfall Model

Waterfall model can be used when

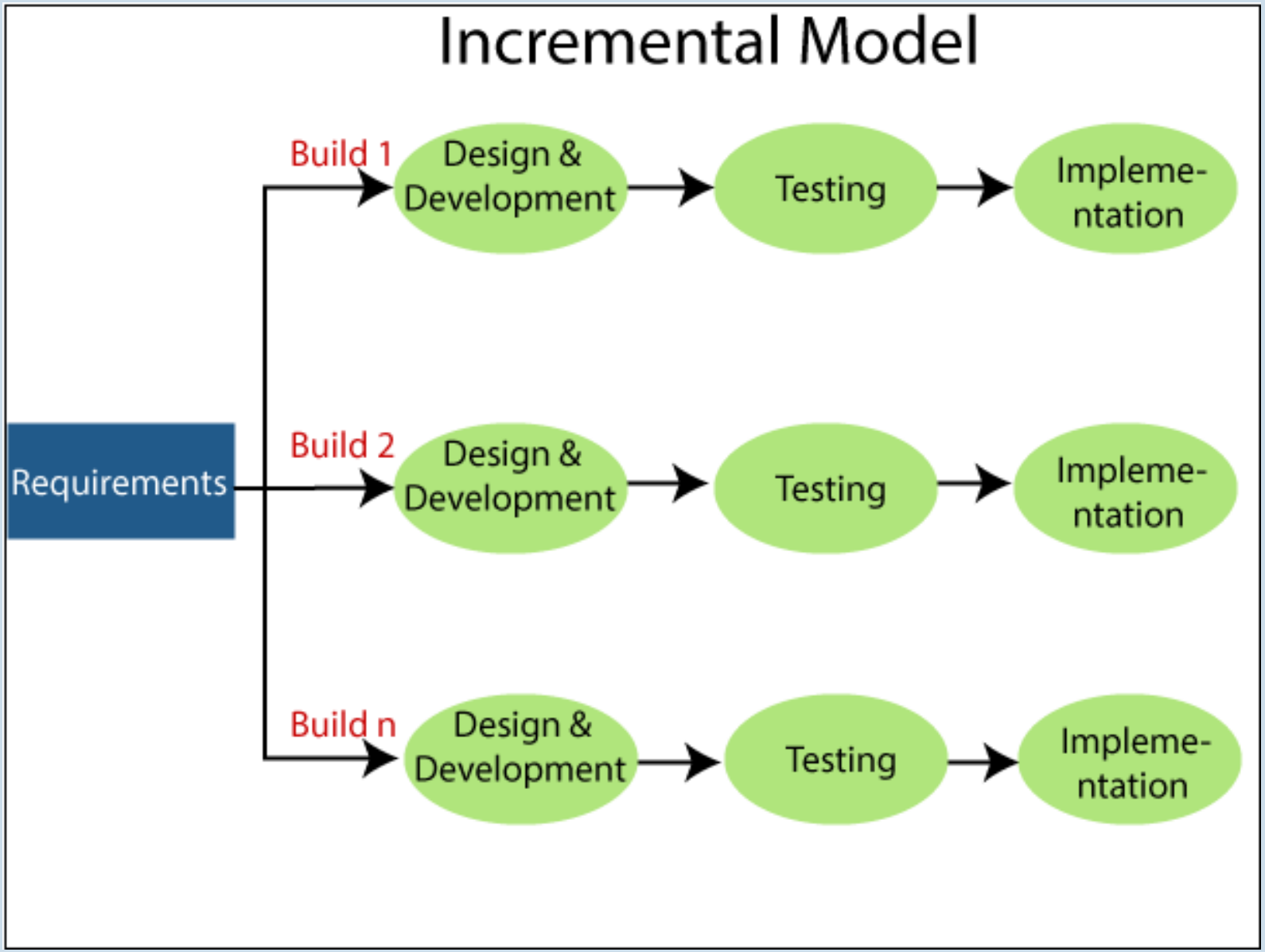
- Requirements are not changing frequently
- Application is not complicated and big
- Project is short
- Requirement is clear
- Environment is stable
- Technology and tools used are not dynamic and is stable
- Resources are available and trained

Advantages and Disadvantages of Waterfall-Model

Advantages	Dis-Advantages
•Before the next phase of development, each phase must be completed	•Error can be fixed only during the phase
•Suited for smaller projects where requirements are well defined	•It is not desirable for complex project where requirement changes frequently
•They should perform quality assurance test (Verification and Validation) before completing each stage	•Testing period comes quite late in the developmental process
•Elaborate documentation is done at every phase of the software's development cycle	•Documentation occupies a lot of time of developers and testers
•Project is completely dependent on project team with minimum client intervention	•Clients valuable feedback cannot be included with ongoing development phase
•Any changes in software is made during the process of the development	•Small changes or errors that arise in the completed software may cause a lot of problems

INCREMENTAL MODEL IN SDLC

- Incremental Model is a software development process where requirements are divided into several stand-alone software development modules.
- In this example, each module passes through the requirement, design, development, implementation, and testing phases.
- That subsequent release of the module adds a feature to the previous release.
- The process will continue until the whole software is achieved.


Incremental Model
Build 1
Design & Development
Testing
Impleme-ntation
Requirements
Build 2
Design & Development
Testing
Impleme-ntation
Build n
Design & Development
Testing
Impleme-ntation

Characteristics of an Incremental module includes

- System development is broken down into many mini development projects
- Partial systems are successively built to produce a final total system
- Highest priority requirement is tackled first
- Once the requirement is developed, requirement for that increment are frozen

- Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.

- Design & Development: In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.

- Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.

- Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental Model

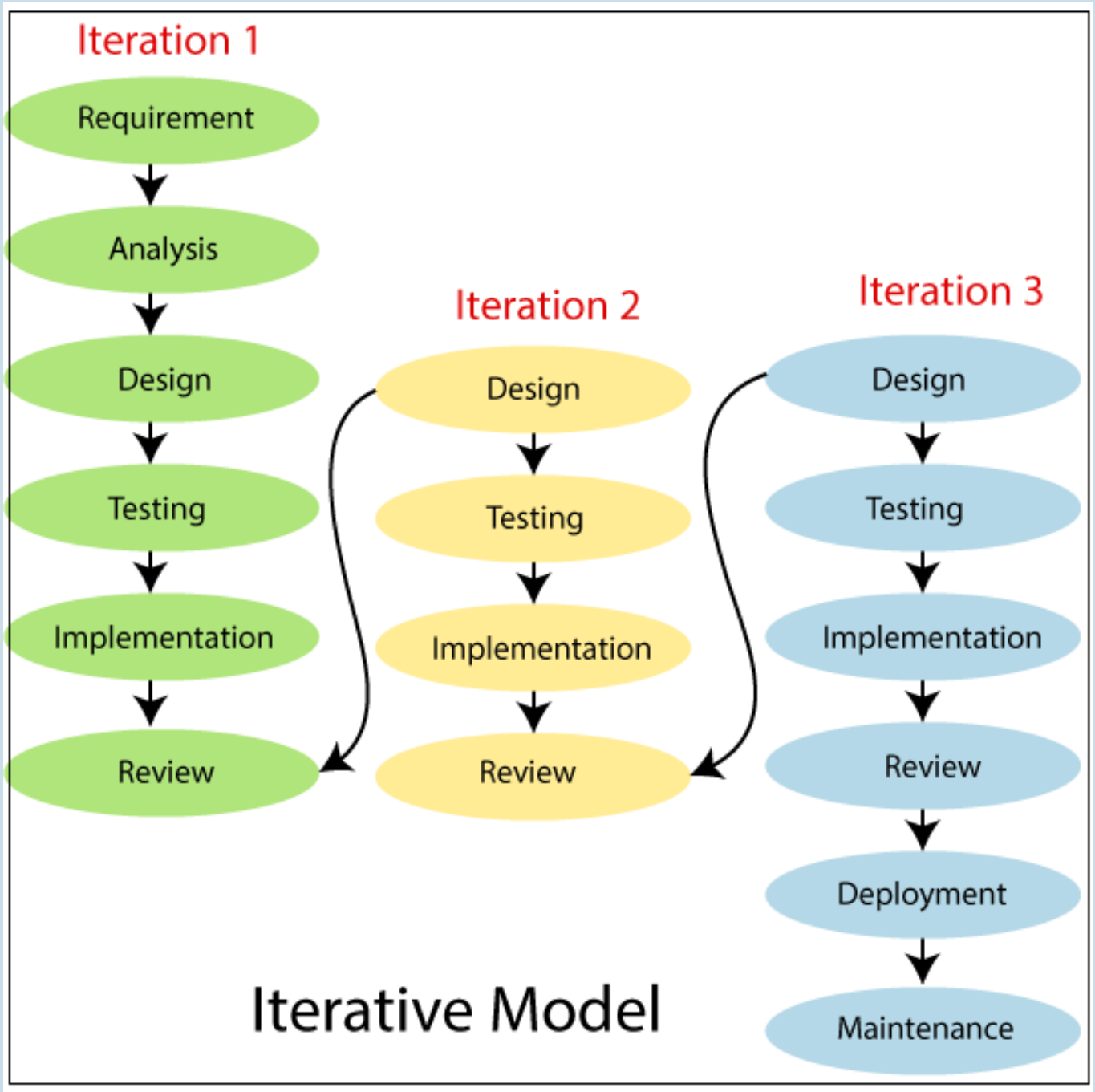
- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Disadvantage of Incremental Model

- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

ITERATIVE MODEL

- This model consists of the same phases as the waterfall model, but with fewer restrictions.
- Generally, the phases occur in the same order as in the waterfall model, but these may be conducted in several cycles.
- A reusable product is released at the end of the cycle, with each release providing additional functionality.


Iteration 1
Requirement
Analysis
Design
Testing
Implementation
Review
Iteration 2
Design
Testing
Implementation
Review
Iteration 3
Design
Testing
Implementation
Review
Deployment
Maintenance
Iterative Model

- Requirement and analysis phase : In this phase, requirements are collected from customers and examined by an analyst to see if the requirements will be met. Analysts examine what should or should not be achieved within the budget. After all this, the software team proceeds to the next stage.

- Design :In the design phase, the team designs the software with different diagrams such as data flow diagram, activity diagram, class diagram, state transition diagram, etc.

- Implementation : In the implementation phase, coding is done, and it is converted to complete software.

- Testing :After completing the coding phase, software testing starts using various testing methods. There are various testing methods, but the most common is white box testing, black-box testing, and gray box testing methods.

- Deployment : After completing all the steps, the software is deployed in its work environment. In this phase, after product deployment, the review phase is carried out to check the behavior and validity of the developed product.

- Review : In this phase, after product deployment, the review phase is carried out to check the behavior and validity of the developed product. And if an error is found, the process begins to review again.

- Maintenance: In the maintenance phase, there may be some bugs after the deployment of the software in the work environment; some errors or new updates are required. Maintenance includes debugging and new additional options.

Advantages of Iterative Model

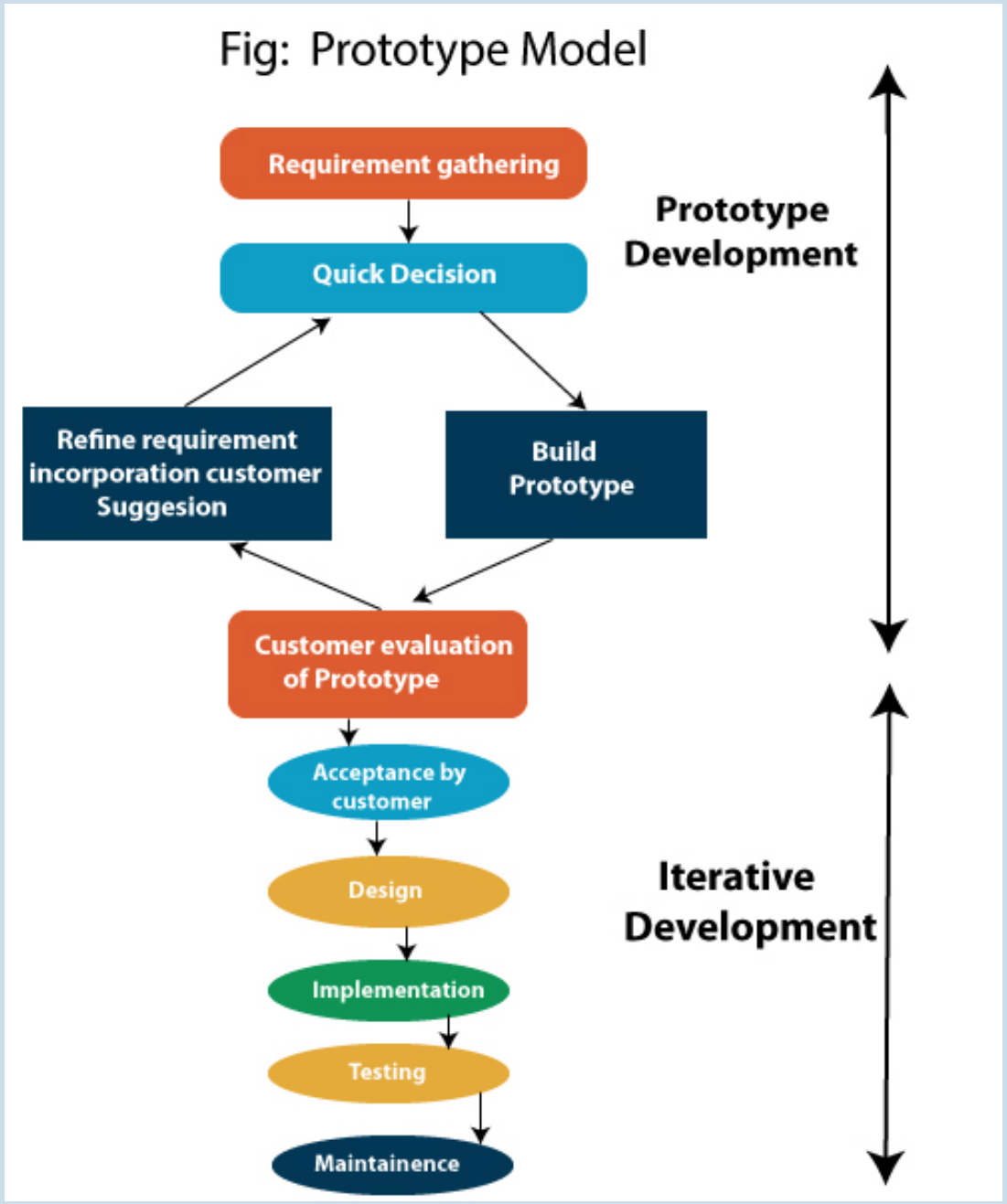
- Many features can be developed quickly in the life cycle.
- Results are received quickly and periodically.
- Testing and debugging is easy during short iterations.
- Less expensive to change scope / requirements.
- Suitable for large projects.

Disadvantages of Iterative Model

- Requires more management attention.
- Risk analysis requires highly efficient resources.
- The project completion date has not been confirmed due to changing requirements.

PROTOTYPE MODEL

- **Prototyping Model** is a software development model in which prototype is built, tested, and reworked until an acceptable prototype is achieved.
- It also creates base to produce the final system or software. It works best in scenarios where the project's requirements are not known in detail.
- It is an iterative, trial and error method which takes place between developer and client.



Steps of Prototype Model

- Requirement Gathering and Analyst
- Quick Decision
- Build a Prototype
- Assessment or User Evaluation
- Prototype Refinement
- Engineer Product

Types of Prototyping Models

Four types of Prototyping models are:

- Rapid Throwaway prototypes
- Evolutionary prototype
- Incremental prototype
- Extreme prototype

Rapid Throwaway Prototype

- Rapid throwaway is based on the preliminary requirement. It is quickly developed to show how the requirement will look visually. The customer's feedback helps drives changes to the requirement, and the prototype is again created until the requirement is baselined.

- In this method, a developed prototype will be discarded and will not be a part of the ultimately accepted prototype. This technique is useful for exploring ideas and getting instant feedback for customer requirements.

Evolutionary Prototyping

Here, the prototype developed is incrementally refined based on customer's feedback until it is finally accepted. It helps you to save time as well as effort. That's because developing a prototype from scratch for every interaction of the process can sometimes be very frustrating.

This model is helpful for a project which uses a new technology that is not well understood. It is also used for a complex project where every functionality must be checked once. It is helpful when the requirement is not stable or not understood clearly at the initial stage.

Incremental Prototyping

In incremental Prototyping, the final product is decimated into different small prototypes and developed individually. Eventually, the different prototypes are merged into a single product. This method is helpful to reduce the feedback time between the user and the application development team.

Extreme Prototyping

Extreme prototyping method is mostly used for web development. It is consists of three sequential phases.

- Basic prototype with all the existing page is present in the HTML format.
- You can simulate data process using a prototype services layer.
- The services are implemented and integrated into the final prototype.

The first phase is the static prototype, consisting of HTML pages and possibly a logical data model supporting those pages. The second phase is a coding process in your chosen web framework whereby the screens are fully functional using a simulated services layer. The third phase is where the services are implemented.

Best practices of Prototyping

Here, are a few things which you should watch for during the prototyping process:

- You should use Prototyping when the requirements are unclear
- It is important to perform planned and controlled Prototyping.
- Regular meetings are vital to keep the project on time and avoid costly delays.
- The users and the designers should be aware of the prototyping issues and pitfalls.
- At a very early stage, you need to approve a prototype and only then allow the team to move to the next step.
- In software prototyping method, you should never be afraid to change earlier decisions if new ideas need to be deployed.
- You should select the appropriate step size for each version.
- Implement important features early on so that if you run out of the time, you still have a worthwhile system

Advantages of the Prototyping Model

- Users are actively involved in development. Therefore, errors can be detected in the initial stage of the software development process.
- Missing functionality can be identified, which helps to reduce the risk of failure as Prototyping is also considered as a risk reduction activity.
- Helps team member to communicate effectively
- Customer satisfaction exists because the customer can feel the product at a very early stage.
- There will be hardly any chance of software rejection.
- Quicker user feedback helps you to achieve better software development solutions.
- Allows the client to compare if the software code matches the software specification.
- It helps you to find out the missing functionality in the system.
- It also identifies the complex or difficult functions.
- Encourages innovation and flexible designing.
- It is a straightforward model, so it is easy to understand.
- No need for specialized experts to build the model
- The prototype serves as a basis for deriving a system specification.
- The prototype helps to gain a better understanding of the customer's needs.
- Prototypes can be changed and even discarded.
- A prototype also serves as the basis for operational specifications.
- Prototypes may offer early training for future users of the software system.

Disadvantages of the Prototyping Model

- Prototyping is a slow and time taking process.
- The cost of developing a prototype is a total waste as the prototype is ultimately thrown away.
- Prototyping may encourage excessive change requests.
- Some times customers may not be willing to participate in the iteration cycle for the longer time duration.
- There may be far too many variations in software requirements when each time the prototype is evaluated by the customer.
- Poor documentation because the requirements of the customers are changing.
- It is very difficult for software developers to accommodate all the changes demanded by the clients.
- After seeing an early prototype model, the customers may think that the actual product will be delivered to him soon.
- The client may lose interest in the final product when he or she is not happy with the initial prototype.
- Developers who want to build prototypes quickly may end up building sub-standard development solutions.

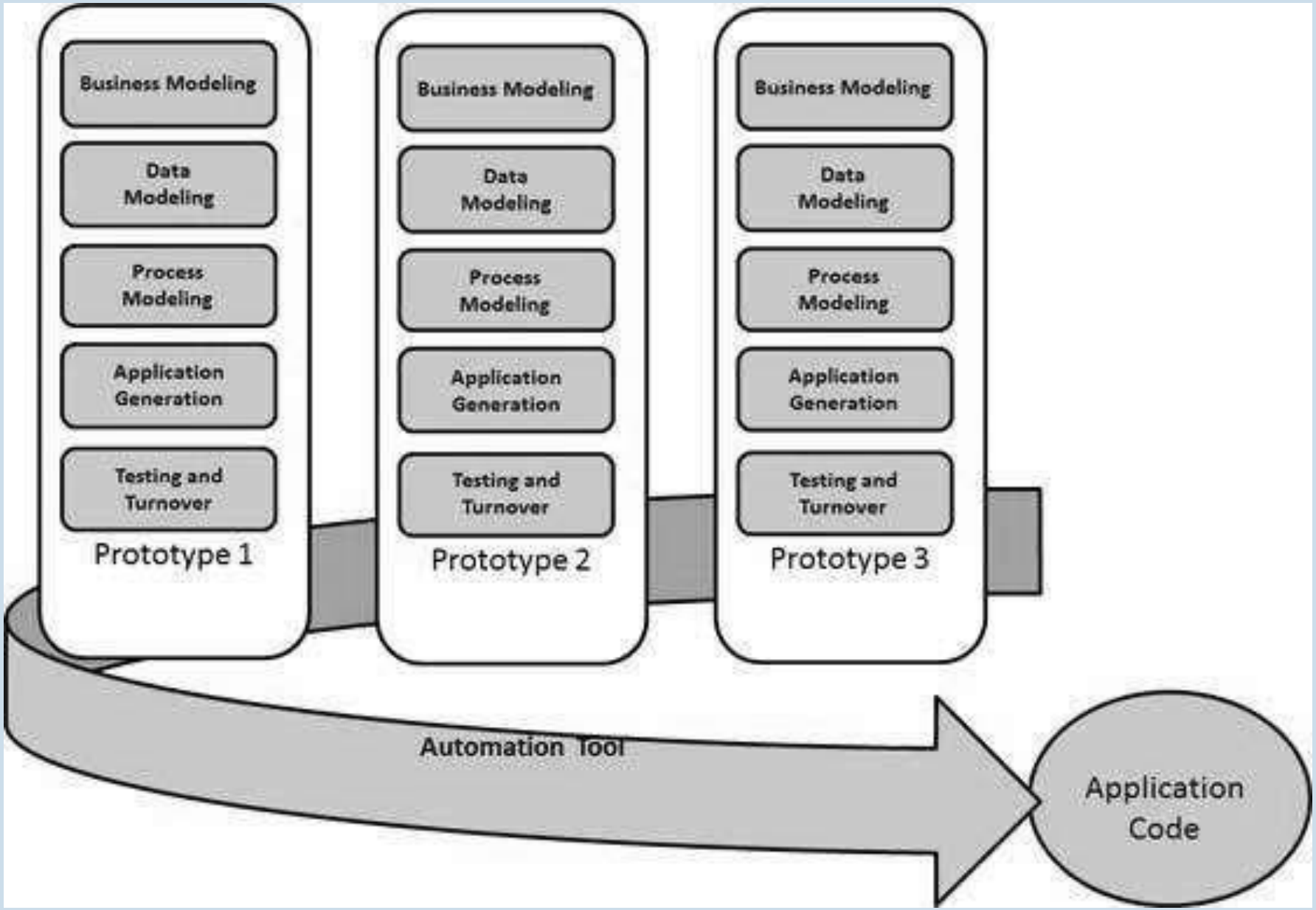
RAD MODEL

- The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved.
- In RAD model, there is less attention paid to the planning and more priority is given to the development tasks.
- It targets at developing software in a short span of time.
- Rapid Application Development 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.

- In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.

- RAD have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.

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


Business Modeling
Data Modeling
Process Modeling
Application Generation
Testing and Turnover
Prototype 1
Business Modeling
Data Modeling
Process Modeling
Application Generation
Testing and Turnover
Prototype 2
Business Modeling
Data Modeling
Process Modeling
Application Generation
Testing and Turnover
Prototype 3
Automation Tool
Application Code

RAD Model Phases	Activities performed in RAD Modeling
Business Modeling	•On basis of the flow of information and distribution between various business channels, the product is designed
Data Modeling	•The information collected from business modeling is refined into a set of data objects that are significant for the business
Process Modeling	•The data object that is declared in the data modeling phase is transformed to achieve the information flow necessary to implement a business function
Application Generation	•Automated tools are used for the construction of the software, to convert process and data models into prototypes
Testing and Turnover	•As prototypes are individually tested during every iteration, the overall testing time is reduced in RAD.

RAD Model – Application

- RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail.
- The following pointers describe the typical scenarios where RAD can be used −
- RAD should be used only when a system can be modularized to be delivered in an incremental manner.
- It should be used if there is a high availability of designers for Modelling.
- It should be used only if the budget permits use of automated code generating tools.
- RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge.
- Should be used where the requirements change during the project and working prototypes are to be presented to customer in small iterations of 2-3 months.

The advantages of the RAD Model are as follows −

- Changing requirements can be accommodated.
- Progress can be measured.
- Iteration time can be short with use of powerful RAD tools.
- Productivity with fewer people in a short time.
- Reduced development time.
- Increases reusability of components.
- Quick initial reviews occur.
- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.

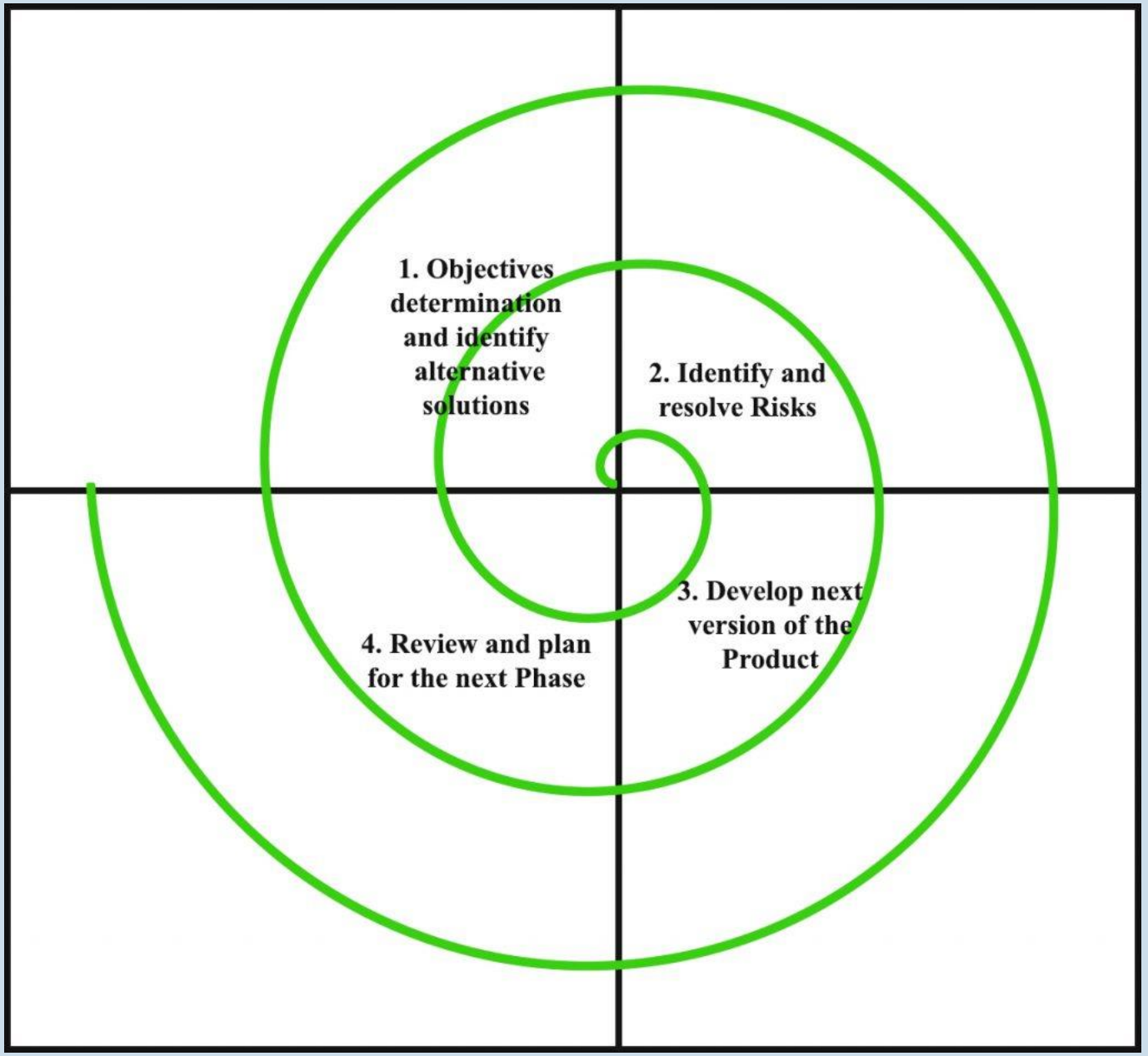
The disadvantages of the RAD Model are as follows −

- Dependency on technically strong team members for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on Modelling skills.
- Inapplicable to cheaper projects as cost of Modelling and automated code generation is very high.
- Management complexity is more.
- Suitable for systems that are component based and scalable.
- Requires user involvement throughout the life cycle.
- Suitable for project requiring shorter development times

SPIRAL MODEL

- Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling.
- In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project.
- Each loop of the spiral is called a Phase of the software development process.
- The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks
- As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using spiral model.

- The Radius of the spiral at any point represents the expenses(cost) of the project so far.
- The angular dimension represents the progress made so far in the current phase.



- Objectives determination and identify alternative solutions: Requirements are gathered from the customers and the objectives are identified, elaborated and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.
- Identify and resolve Risks: During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution is identified and the risks are resolved using the best possible strategy.
- Develop next version of the Product: During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.
- Review and plan for the next Phase: In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

RISK HANDLING IN SPIRAL MODEL

- A risk is any adverse situation that might affect the successful completion of a software project.
- The most important feature of the spiral model is handling these unknown risks after the project has started. Such risk resolutions are easier done by developing a prototype.
- The spiral model supports coping up with risks by providing the scope to build a prototype at every phase of the software development.
- Prototyping Model also support risk handling, but the risks must be identified completely before the start of the development work of the project.
- But in real life project risk may occur after the development work starts, in that case, we cannot use Prototyping Model.
- In each phase of the Spiral Model, the features of the product dated and analyzed and the risks at that point of time are identified and are resolved through prototyping. Thus, this model is much more flexible compared to other SDLC models.

WHY SPIRAL MODEL IS CALLED META MODEL ?

- The Spiral model is called as a Meta Model because it subsumes all the other SDLC models.
- For example, a single loop spiral actually represents the Iterative Waterfall Model.
- The spiral model incorporates the stepwise approach of the Classical Waterfall Model.
- The spiral model uses the approach of Prototyping Model by building a prototype at the start of each phase as a risk handling technique.
- Also, the spiral model can be considered as supporting the evolutionary model – the iterations along the spiral can be considered as evolutionary levels through which the complete system is built.

ADVANTAGES OF SPIRAL MODEL:

- Risk Handling: The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
- Good for large projects: It is recommended to use the Spiral Model in large and complex projects.
- Flexibility in Requirements: Change requests in the Requirements at later phase can be incorporated accurately by using this model.
- Customer Satisfaction: Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

DISADVANTAGES OF SPIRAL MODEL:

- Complex: The Spiral Model is much more complex than other SDLC models.
- Expensive: Spiral Model is not suitable for small projects as it is expensive.
- Too much dependable on Risk Analysis: The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced expertise, it is going to be a failure to develop a project using this model.
- Difficulty in time management: As the number of phases is unknown at the start of the project, so time estimation is very difficult.

AGILE METHODOLOGY

- AGILE methodology is a practice that promotes continuous iteration of development and testing throughout the software development lifecycle of the project.
- In the Agile model, both development and testing activities are concurrent, unlike the Waterfall model.
- The Agile software development methodology is one of the simplest and effective processes to turn a vision for a business need into software solutions.
- Agile is a term used to describe software development approaches that employ continual planning, learning, improvement, team collaboration, evolutionary development, and early delivery. It encourages flexible responses to change.
- MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT : https://agilemanifesto.org/

- The Agile model was primarily designed to help a project to adapt to change requests quickly. So, the main aim of the Agile model is to facilitate quick project completion. To accomplish this task agility is required.
- Agility is achieved by fitting the process to the project, removing activities that may not be essential for a specific project. Also, anything that is wastage of time and effort is avoided.
- Agile model is the combination of iterative and incremental process models.
- The time to complete an iteration is known as a Time Box.
- Time-box refers to the maximum amount of time needed to deliver an iteration to customers.
- So, the end date for an iteration does not change. Though the development team can decide to reduce the delivered functionality during a Time-box if necessary to deliver it on time.
- The central principle of the Agile model is the delivery of an increment to the customer after each Time-box.

WHAT ARE THE 4 AGILE VALUES?

Agile Values refers to the set of 4 values outlined by the Agile Alliance in The Agile Manifesto. This set of values encourages putting people before processes, getting software out the door fast, collaborating with customers, and adjusting plans as needed.

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

INDIVIDUALS AND INTERACTIONS OVER PROCESSES AND TOOLS

- No matter how well-researched your process and high-tech your tools are, it's the team you work with and the way you work together that determines success. Your team and their ability to communicate effectively and efficiently is more valuable than the processes they follow or the tools you use.

- This isn't to say that agile philosophies discourage formalized processes or tools. Both can be helpful in providing structure for your team and facilitating interactions. But at the end of the day, they come second.

- After all, processes and tools are worthless if your team can't communicate. But put a smart, motivated team up to a task without any processes or tools to manage the project and chances are they'll find some way to get it done.

WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTATION

- Traditional product development processes often required extensive documentation before a single line of code was written. Under the agile philosophy, getting software in the hands of customers is the highest priority. After all, how are you going to improve your product if you don't get it out in the wild and collect feedback from real users?

- While this value highlights the importance of shipping software over letting documentation be a bottleneck, it's important to note that documentation in itself is not a bad thing…as long as you don't overdo it.

CUSTOMER COLLABORATION OVER CONTRACT NEGOTIATION

- The agile philosophy highlights the importance of customer-centric product development practices over product-centric approaches. While contracts will always have their place in business, a list of the things you're offering your customer is no replacement for actually communicating with them about what their needs are and where their challenges are.

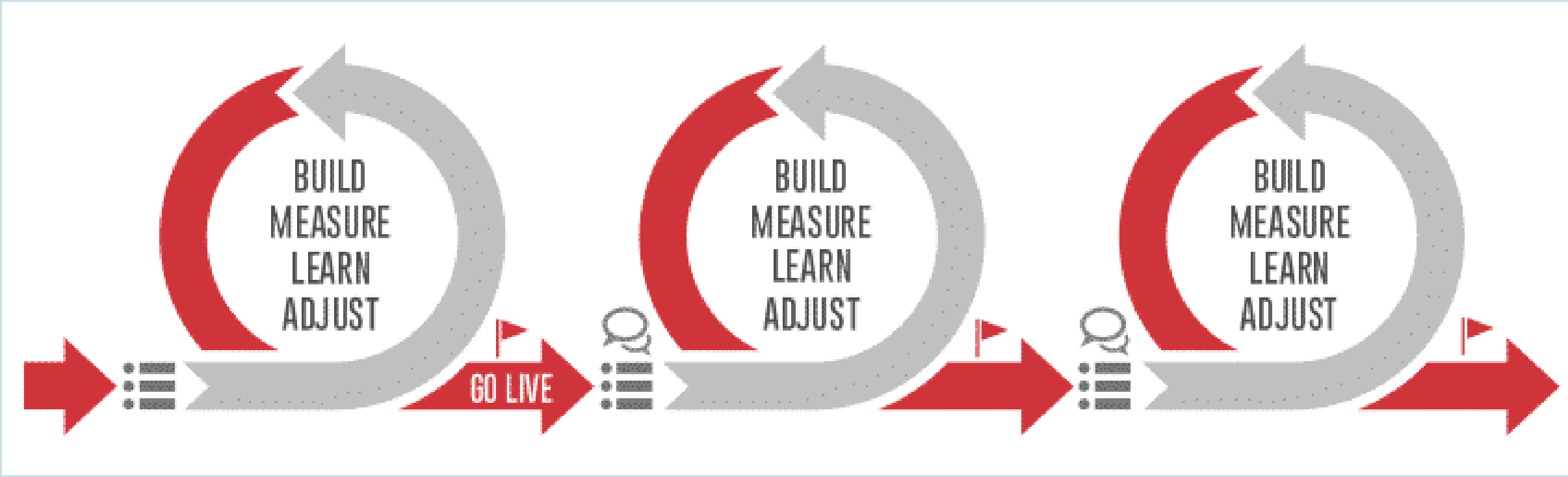
- Traditional product-centric processes allowed contracts to dictate what was delivered in the end, which left a lot of room for mismatched expectations. The agile philosophy (and many of the formalized processes that have come out of it) encourage building a continuous customer feedback loop into development cycles.

- Under the agile philosophy, customer collaboration begins early in the development process and happens frequently throughout. This culture of close collaboration with real customers helps product people ensure they're delivering effective, useful solutions to customers. When you talk to customers often and build feedback into your development process, you reduce risk and eliminate guesswork.

RESPONDING TO CHANGE OVER FOLLOWING A PLAN

- An important benefit of the agile methodology is that it encourages frequent reviewing and retooling of current plans based on new information that the team is continually gathering and analyzing.
- The product roadmap, then, is no longer a static document, but a dynamic strategy. Product managers in agile environments will need to learn to present their dynamic roadmaps to stakeholders in a transparent manner that reflects the likelihood of change based on new learnings.

- In other words, the agile methodology lets a product team adjust its priorities and plans whenever doing so makes strategic sense. These teams do not get stuck in an outdated plan simply because they have committed to seeing it through.


BUILD
MEASURE
LEARN
ADJUST
GO LIVE
BUILD
MEASURE
LEARN
ADJUST
BUILD
MEASURE
LEARN
ADJUST

Remember: Agile Values are not Rules

Finally, don't forget that agile is a mindset, not a set of strict rules to follow. These values are up for interpretation and are not solid instructions set in stone.

PRINCIPLES OF AGILE MODEL:

- To establish close contact with the customer during development and to gain a clear understanding of various requirements, each Agile project usually includes a customer representative on the team. At the end of each iteration stakeholders and the customer representative review, the progress made and re-evaluate the requirements.
- Agile model relies on working software deployment rather than comprehensive documentation.
- Frequent delivery of incremental versions of the software to the customer representative in intervals of few weeks.
- Requirement change requests from the customer are encouraged and efficiently incorporated.
- It emphasizes on having efficient team members and enhancing communications among them is given more importance. It is realized that enhanced communication among the development team members can be achieved through face-to-face communication rather than through the exchange of formal documents.
- It is recommended that the development team size should be kept small (5 to 9 peoples) to help the team members meaningfully engage in face-to-face communication and have collaborative work environment.
- Agile development process usually deploy Pair Programming. In Pair programming, two programmers work together at one work-station. One does coding while the other reviews the code as it is typed in. The two programmers switch their roles every hour or so.

Here are lists of different kinds of Agile SDLC models that are used by software development companies. These are:

- Feature-driven development
- Crystal
- Atern
- Scrum
- Unified process
- Extreme programming (XP)
- Lean development

Amongst them, Scrum, Lean and Extreme programming are some of the most popular forms of Agile development methodologies.

Classical waterfall	Iterative waterfall	Prototype model	Incremental model	Evolutionary model	RAD model	Spiral model	Agile model
Basic, Rigid, Inflexible, Not for real project	Basic, problem is well understood	User requirement not clear, costly, no early lock on requirements, high user involvement, reusability	Module by module delivery, easy to test and debug	Large projects	Time and cost constraint, user at all levels, reusability	Risk, not for small projects, no early lock on requirements	Flexible, advanced , parallel, process divided into sprints

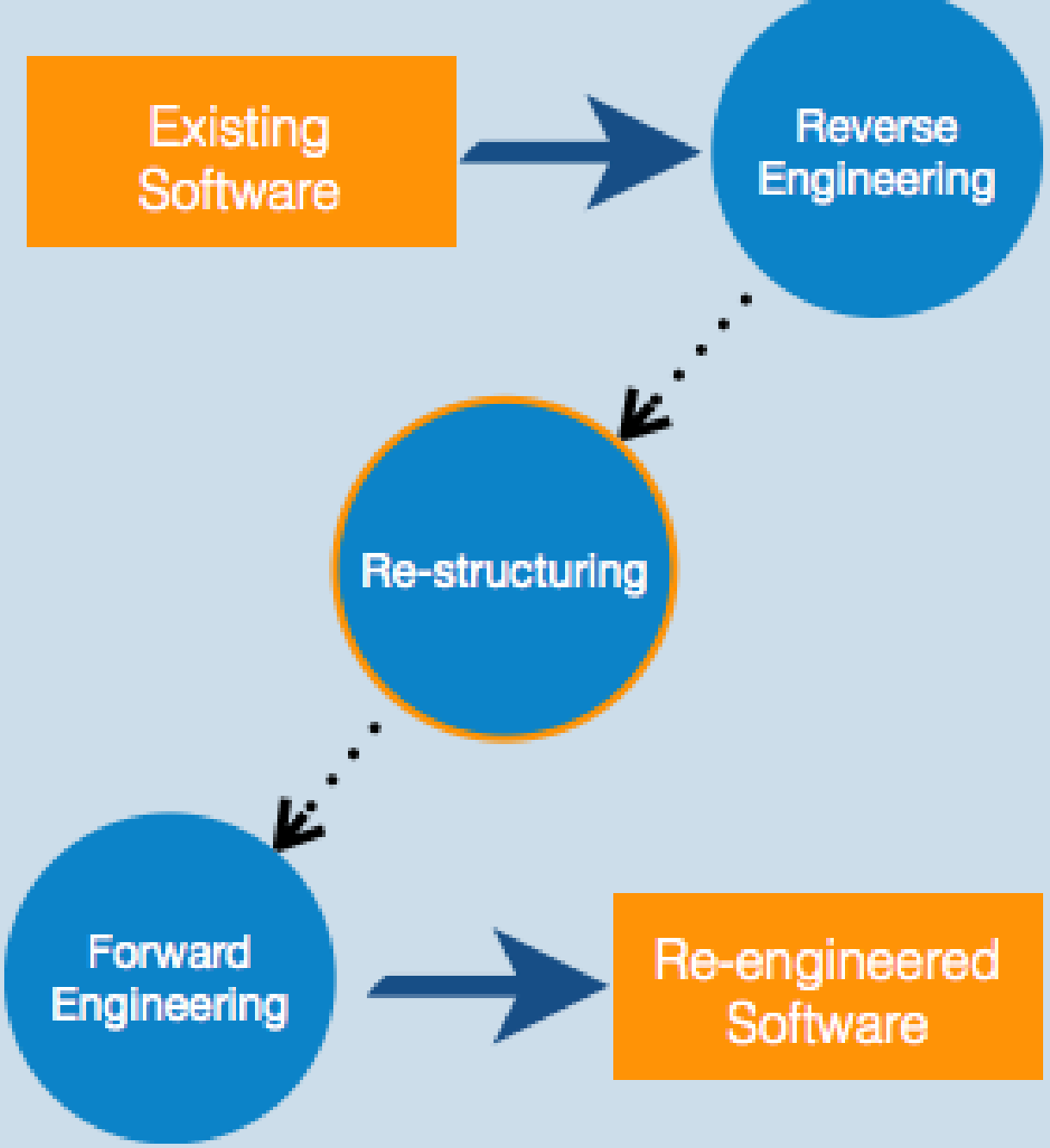
SOFTWARE RE-ENGINEERING

- When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

- Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

- For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

- Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.


Existing Software
Reverse Engineering
Re-structuring
Forward Engineering
Re-engineered Software

RE-ENGINEERING PROCESS

- Decide what to re-engineer. Is it whole software or a part of it?
- Perform Reverse Engineering, in order to obtain specifications of existing software.
- Restructure Program if required. For example, changing function-oriented programs into object-oriented programs.
- Re-structure data as required.
- Apply Forward engineering concepts in order to get re-engineered software.

REVERSE ENGINEERING

- It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

- An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.

- The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

REVERSE ENGINEERING GOALS:

- Cope with Complexity.
- Recover lost information.
- Detect side effects.
- Synthesise higher abstraction.
- Facilitate Reuse.

DIFFERENCE BETWEEN FORWARD ENGINEERING AND REVERSE ENGINEERING

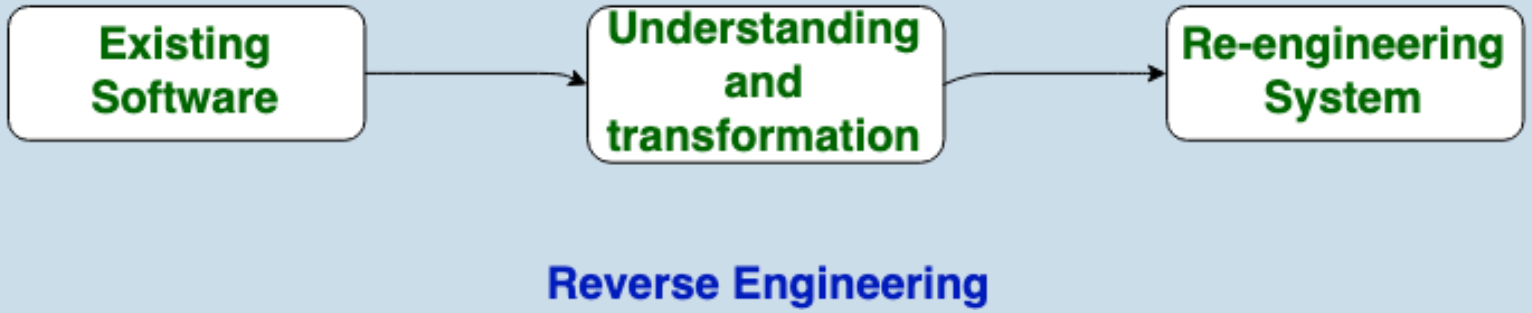
Forward Engineering:

Forward Engineering is a method of creating or making an application with the help of the given requirements. Forward engineering is also known as Renovation and Reclamation. Forward engineering is required high proficiency skill. It takes more time to construct or develop an application.

Forward Engineering

Reverse Engineering:

Reverse Engineering is also known as backward engineering, is the process of forward engineering in reverse. In this, the information are collected from the given or exist application. It takes less time than forward engineering to develop an application. In reverse engineering the application are broken to extract knowledge or its architecture.



Reverse Engineering

Forward Engineering	Reverse Engineering
In forward engineering, the application are developed with the given requirements.	In reverse engineering or backward engineering, the information are collected from the given application.
Forward Engineering is high proficiency skill.	Reverse Engineering or backward engineering is low proficiency skill.
Forward Engineering takes more time to develop an application.	While Reverse Engineering or backward engineering takes less time to develop an application.
The nature of forward engineering is Prescriptive.	The nature of reverse engineering or backward engineering is Adaptive.
In forward engineering, production is started with given requirements.	In reverse engineering, production is started by taking existing product.