# Introduction to Software Engineering (4340702)

Computer Department
AVPTI Rajkot

# Teaching and Examination Scheme

| Theory Marks | | Practical Marks | | Total Marks |
|---|---|---|---|---|
| ESE | CA | ESE | CA | **150** |
| 70 | 30 | 25 | 25 | |

| Lecture | Tutorial | Practical | Credit |
|---|---|---|---|
| 3 | 0 | 2 | 4 |

# COURSE OUTCOMES (COs)

- Compare various software development process models.
- Prepare software analysis and design using SRS, DFD and object oriented UML diagrams.
- Prepare software development plan using project scheduling.
- Prepare test-cases to test software functionalities.

# Unit – I
# Software Process Models

# Unit Outcomes

- Define Software Engineering.
- Recommend the relevant software solution for the given problem.
- Describe Generic Framework Activity
- Select the relevant software process model for the given problem statement with justification.
- Suggest the relevant activities in Agile Development Process in the given situation with justification

# Software

- It is a collection of computer programs,procedures,rules and associated documentation and data.
- Types of software

1. **System Software**

operate the computer hardware , to provide basic functionality needed by users and other software, and to provide a platform for running application software

- **Operating systems** (OS): Windows, Linux, macOS, etc.
- **Device drivers**: software that enables the communication between hardware and OS.
- **Utility software**: tools for system maintenance and optimization.
- **Boot loaders**: software that initializes the OS during startup.

2. **Application Software**

which uses the computer system to perform special functions.

- Office, Excel, Word, PowerPoint, Outlook, etc.
- Firefox, Chrome, Safari, Internet Explorer.

# Continue…

## 3. Embedded Software

it is a piece of software that is embedded in hardware or non-PC devices. It is written specifically for the particular hardware that it runs on.

- Smartwatches, Fitness Trackers, Home Automation Systems, Air Conditioning Systems, Medical Devices etc.

## 4. Web Application

it is an application program that is stored on a remote server and delivered over the internet through a browser interface.

- Google apps,E-commerce,Social Media , Banking etc.

## 5. Artificial Intelligence Software

it is a computer program which mimics human behavior by learning various data patterns and insights.

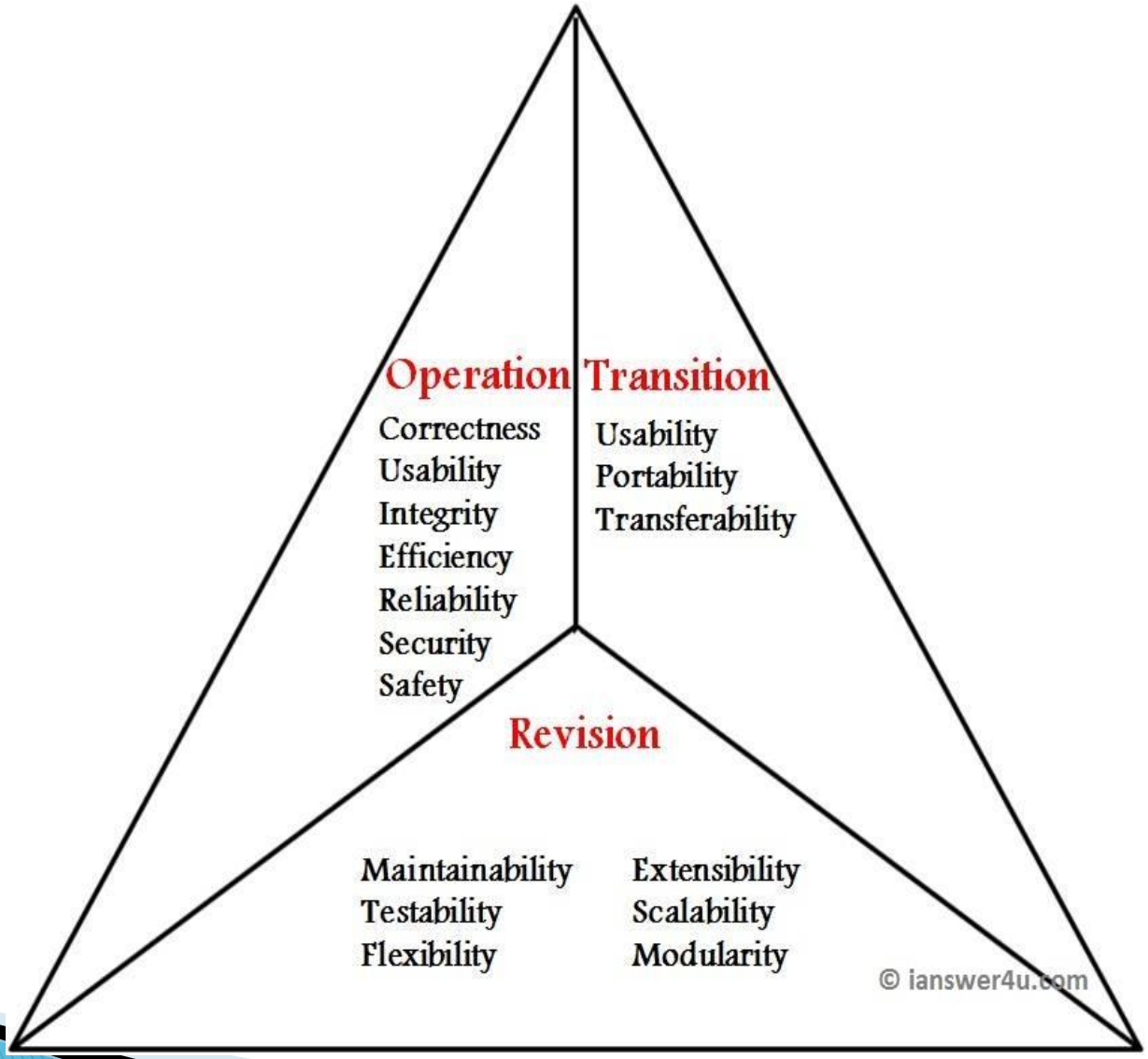
- Google Assistant , ChatGPT , Amazon Alexa etc.

# Program V/S Software

- Developed by individual
- Small in size
- Limited functionality
- Programmer himself is the only user
- Little documentation
- User interface may not very important
- Develop using programmer's individual style

- Large number of Developers
- Very large in size
- Multiple functionality
- Developers and users are totally different
- Large documentation
- User interface may very important
- Develop using Software engineering principle

# Characteristics of software



# Operational Characteristics

a) **Correctness**: The software which we are making should meet all the specifications stated by the customer.

b) **Usability/Learnability**: The amount of efforts or time required to learn how to use the software should be less. This makes the software user-friendly .

c) **Integrity** : quality software should not have side effects.

d) **Reliability** : The software product should not have any defects. Not only this, it shouldn't fail while execution.

e) **Efficiency** : The software should make effective use of the storage space and execute command as per desired timing requirements.

f) **Security** : The software shouldn't have side effects on data / hardware. Proper measures should be taken to keep data secure from external threats.

g) **Safety** : The software should safe.

# Revision Characteristics

a) **Maintainability** : Maintenance of the software should be easy for any kind of user.

b) **Flexibility** : Changes in the software should be easy to make.

c) **Extensibility** : It should be easy to increase the functions performed by it.

d) **Scalability** : It should be very easy to upgrade it for more work(or for more number of users).

e) **Testability** : Testing the software should be easy.

f) **Modularity** : Software must be divided in to separate individual working modules or parts.

# Transition Characteristics

a) **Interoperability** : Interoperability is the ability of software to exchange information with other applications and make use of information transparently.

b) **Reusability** : If we are able to use the software code with some modifications for different purpose then we call software to be reusable.

c)**Portability** : The ability of software to perform same functions across all environments and platforms, demonstrate its portability.

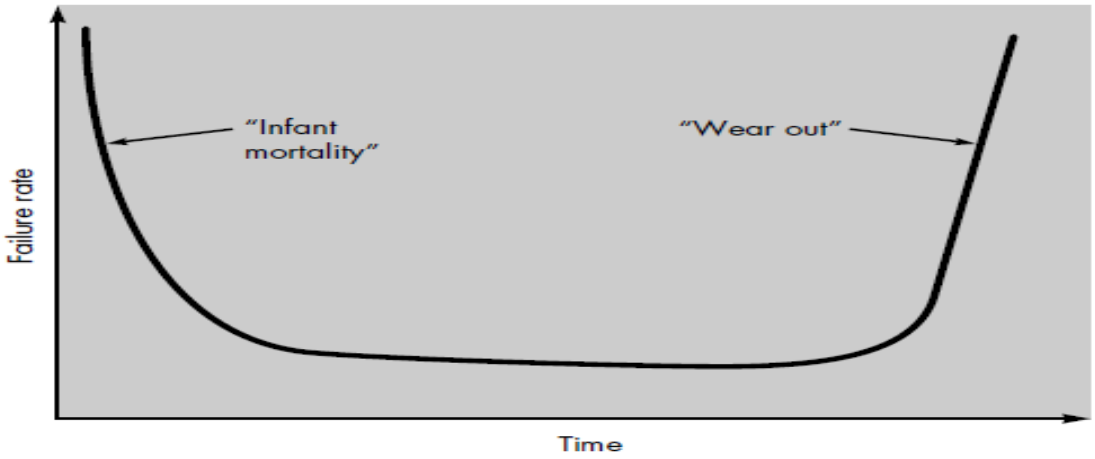
# Software is engineered , not manufactured like hardware.

- Once a product is manufactured, it is not easy to modify it, change. While in case of software we can easily change or modify or change it for later use.

  Even making multiple copies of software is a very easy.
- In hardware, costing is due to assembly of raw material and other processing expenses while in software development no assembly needed like hardware. Hence, software is not manufactured as it is developed or it is engineered.

# Software does not wear out

- Hardware can damage after running time. It can be affected by environmental effects. So the failure rate rises.

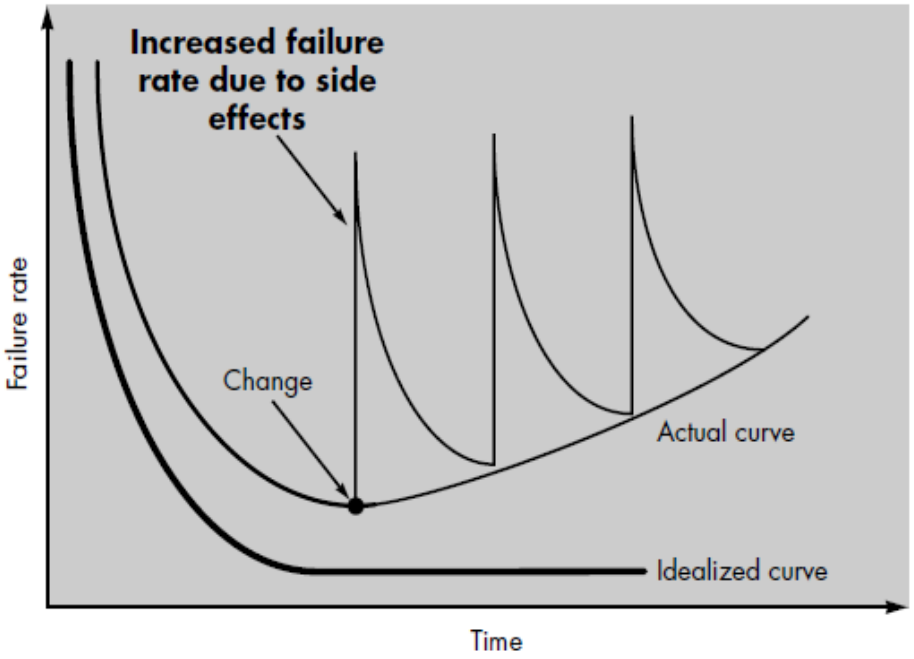



H/W failure curve

- "**bathtub curve**" shows hardware failure
- there are three phases in h/w life
- initially failure rate is much more. But after testing and defects are corrected, failure rate come down.
- In it, h/w is much more useful and chance of failure is quite low.

# Continue…

- As time passes, however, the failure rate rises again as hardware components suffer from the affects of dust, vibration, abuse, temperature extremes, and many other environmental factors.
- So simply, hardware does wear out




S/W failure curve

# Continue…

- Software is not highly affected by environmental effects. The “idealized curve” shows software failure.
- In the early stage, due to lot many errors, software could have high failure. But it becomes reliable as time passes instead of wearing out. Software become reliable.
- Software may be retired due to new requirements, new expectations etc.
- Hence, software doesn’t wear out, but it may be deteriorate.

# Software gives component based construction, it gives reusability of components.

- All large software divided in to some modules or components.
- All this module developed individually and than integrated to develop a complete software.
- All these module can be used for other similar types of software.
- So software dives reusability of components.

# Software is flexible for custom built.

- A software can be developed to do many types of functions.
- Any kind of change needed in software easily done.
- A software or product can be built on user requirements basis or custom built.

# INTRODUCTION

- **Software engineering** is an engineering approach for software development
- A small program can be written without SE principles.
- To develop large software with good quality and cost effective we have to use SE principles.

# Various Definition:

- **SE is an engineering discipline that covers all aspects of s/w from specification to maintenance.**
- **SE is an engineering discipline that delivers high quality s/w at agreed cost & in planed schedule.**
- SE provide framework that guides the s/w engineers to develop the software.
- SE tells how s/w will work with machines.
- SE covers technical and management issues.
- Three main aspects of SE is → **(Quality S/W at agreed cost in schedule time)**
  - Provide quality product
  - Expected cost
  - Complete work on agreed schedule

# Continue…

- SE is the establishment and use of sound engineering principles in order to obtain economically s/w that is reliable and work efficiently on real machines.
- **(IEEE Definition)** → "Software engineering is the application of a symmetric , disciplined and quantifiable approach to the development, operation and maintenance of software."
- **(Somerville)**: Software Engineering is concerned with the theories, methods and tools to develop the software products in a cost effective way.

# Necessity of Software Engineering

- Program is like a small wall
- You can build it using your common sense and materials like bricks and cement

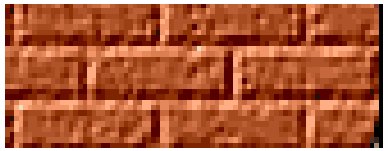


Fig: Small Wall

# Continue…

- Software is like a Large building.
- It is Difficult to Build large building
- You need knowledge of Civil engineering, strength of materials ,testing ,planning ,architectural design etc.
- So the building a small wall and large building are entire different things.

Fig:Large Building

# Continue…

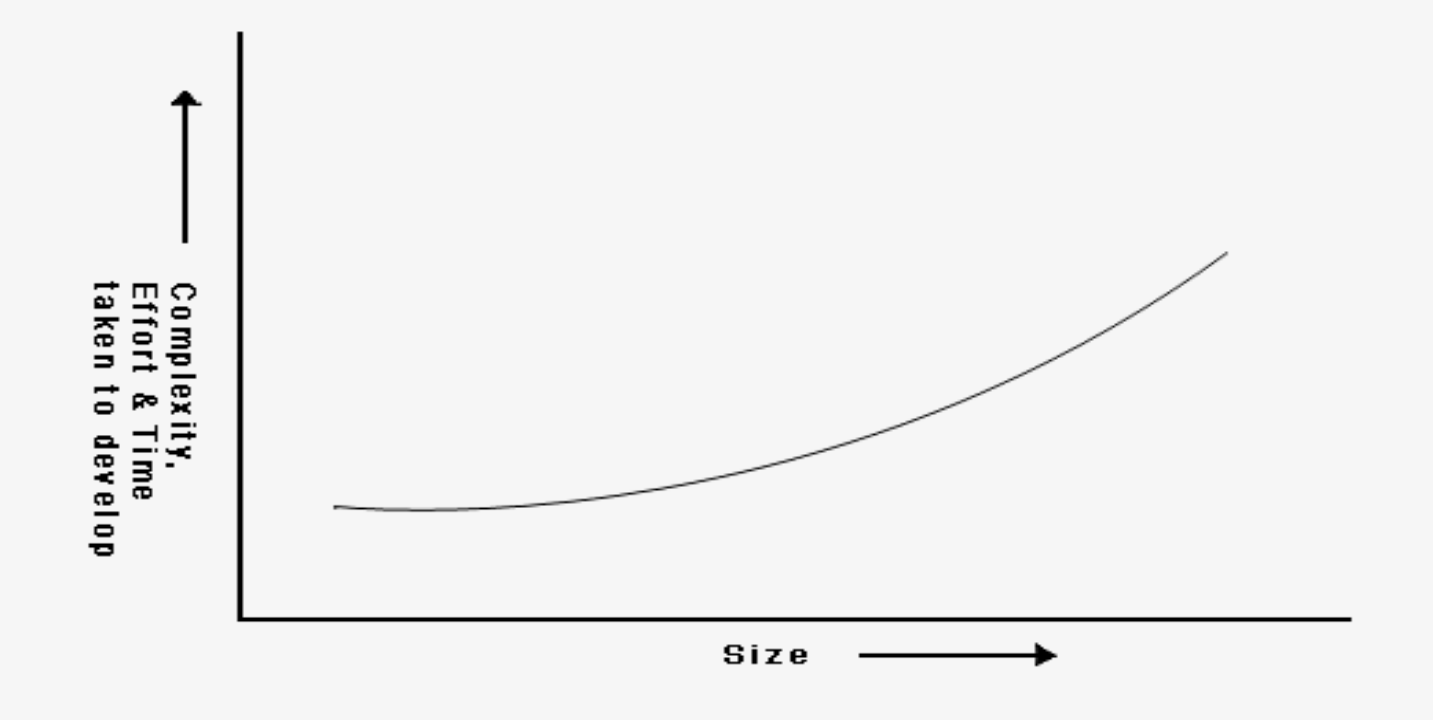


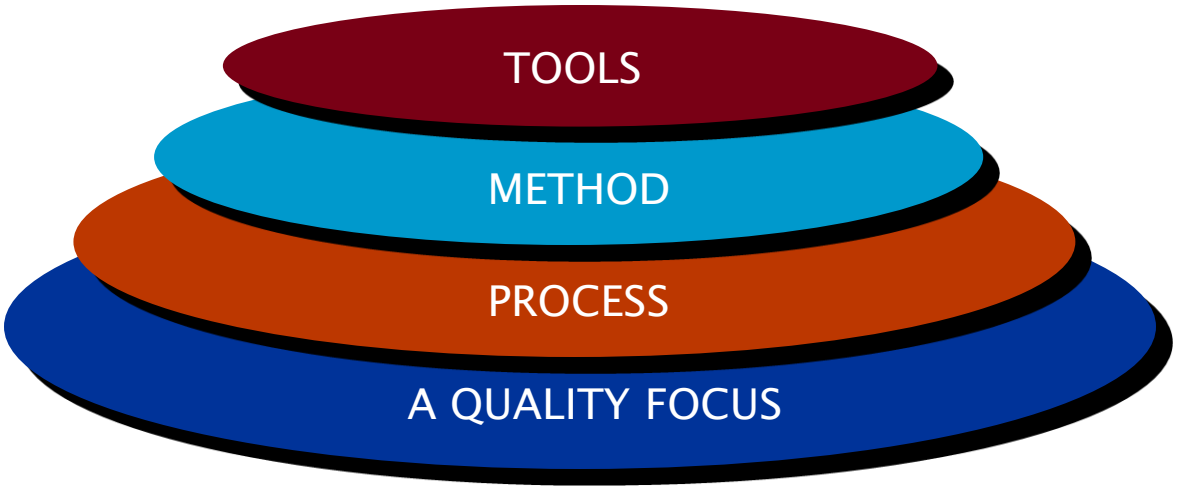

Fig : Increase in Development and effort With Problem Size

# Continue…

- Without SE Principles difficult to develop Large Program
- Complexity and difficulty level increase exponentially with their size as shown in fig.
- Difficulty increase exponentially with LOC (lines of code)
- Increase in LOC 10 times make your program more than 10 time difficult without using SE principles.
- In such situations you have to use SE principle

# Software Engineering Layered approach

- Software engineering can be viewed as a layered technology. Actually software engineering is totally a layered technology.
- It encompasses process, methods, tools that enables a s/w product to be built in a timely manner.
- Four layers are there.
  - Quality
  - Process
  - Method
  - Tools




SE Layers

# Continue…

- **A Quality focus Layer**
  - SE mainly focuses on quality product.
  - It checks whether the output meets with its requirement specifications or not.
  - Every organization should maintain its total quality management.
  - This layer supports software engineering.
- **Process Layer**
  - It is the heart of the SE.
  - It is a foundation layer for development.
  - s/w process is a set of activities together if ordered and performed properly, the desired result would be produced.
  - Define framework activities.
  - Main idea → ***is to deliver s/w in a timely manner.***

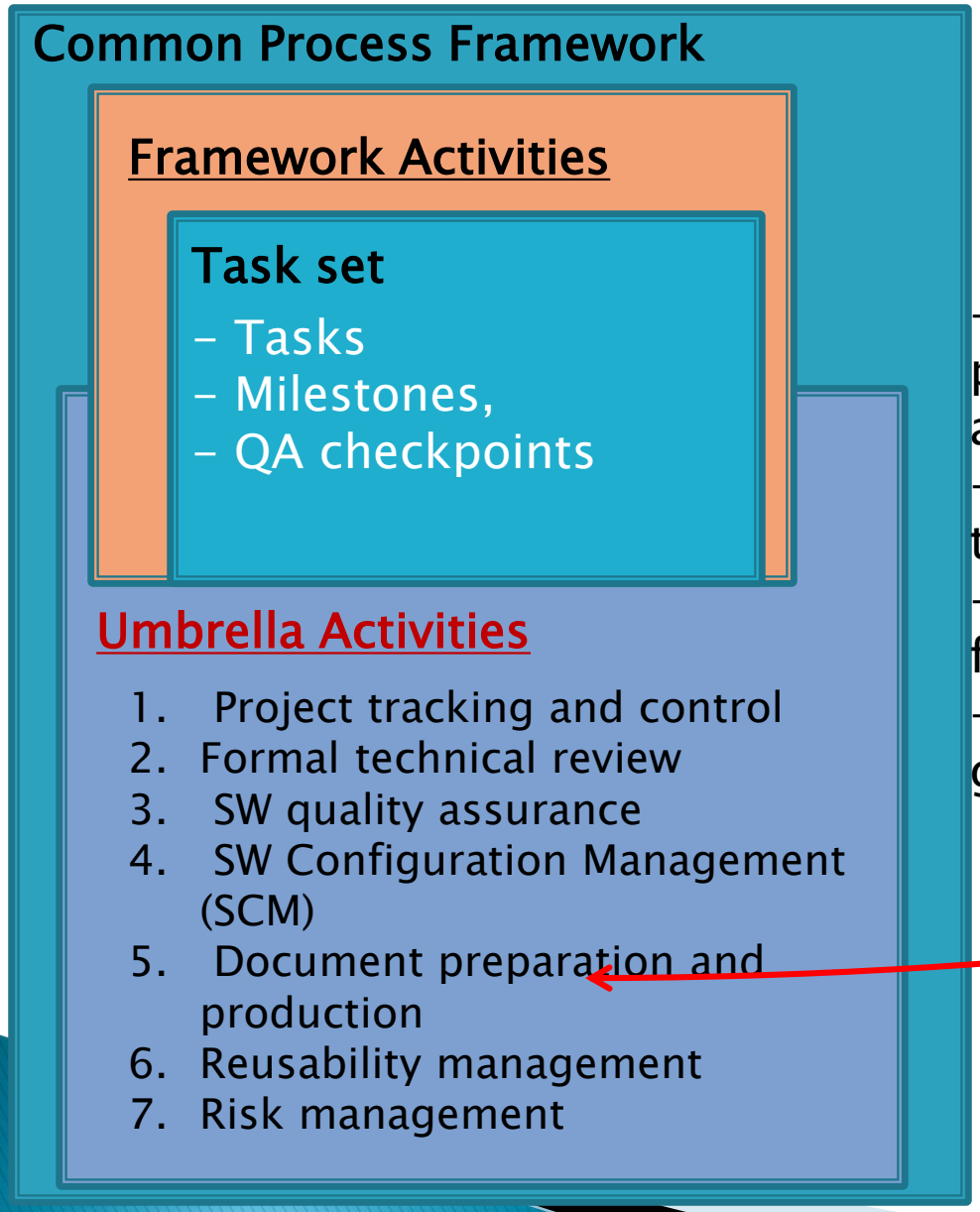
# Continue…

- **Method Layer**
  - It describes ‘how-to’ build software product.
- **Tools layer**
  - It provides different types of tools for software development.
  - Execute process in proper manner.

# Software Development

- Software development is the process of developing software through successive phases in an orderly way.
- This process includes not only the actual writing of code but also the preparation of requirements and objectives, the design of what is to be coded, and confirmation that what is developed has met objectives.
- In other words, Software development is the analysis,computer programming, documenting, testing and bug fixing involved in creating and maintaining application.

# Generic Framework and Umbrella activities




- Each framework activity is populated by set of task, milestones and quality assurance.
- Umbrella activities are performed through out the process.
- these are independent of any framework activity.
- the list of umbrella activities are given in the figure.

# Software Development Life Cycle (SDLC)

- A **software life cycle** is series of identifiable stages that software undergoes during its lifetime.
- The first stage is Feasibility study than requirement analysis and specification ,design ,coding ,testing and maintance.
- Each of these stages called a life cycle phases.
- A **Software Development Life Cycle Model** is a descriptive and diagrammatic representation of the software life cycle.
- A life cycle model represents all the activities(in order) required to make a software product.(inception to retirement)
- It is also known as a **software process model.**

# Why we use a life cycle model ?

- It is used in all modern software development organizations.
- It describes all activities in systematic and disciplined manner.
- s/w is develop by team so all members must know when to do what otherwise it will lead to project failure.
- For example if s/w is divide in several parts and assign work to the team members and than give freedom to them to do this work.

# Continued……

- It is possible that one member might start coding ,another might start to prepare document and some other might start with design.
- So at the end it is difficult integrate this parts and manage overall development.
- It is the main reason of many project failure in past.
- So SDLC must be used to develop a software.

# Why Document a Life Cycle Model?

- software development organizations normally prepare accurate document of the life cycle model which they used.
- It helps to avoid misinterpretations and also helps in identifying the inconsistencies and redundancies.
- With help of this document developers can easily understand the process of development.
- It is also indicate the quality of software so if software development organizations is not using document lifecycle model than that organizations is not capable of developing good quality software.

# Phase entry and exit Criteria

- A life cycle model defines the entry and exit criteria of every phase.
- A phase can begin only when phase entry criteria is satisfied and it consider to be complete only when the exit criteria is satisfied.
- For example The phase entry criteria for software requirement specification phase is software requirement specification SRS document has been developed and approved by the customer. when these criteria are satisfied than only next phase can start.

# Continued……

- If these criteria is well defined it becomes easier to monitor the progress of the project.
- If no clear specification about these criteria than it becomes very difficult to chart the progress of the project.
- This usually leads to a problem that is known as a **99% complete syndrome .**it occurs when there is no definite way to assess the progress of project.
- Here the team members feel that project is 99% complete but actually the project is far from its completion and this make the project completion time highly inaccurate.

# 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:
- Classical Waterfall Model
- Iterative Waterfall Model
- Prototyping Model
- Evolutionary Model
- Spiral Model
- RAD Model

# Different phases of the classical waterfall model

- The classical waterfall model is basic model to develop software.

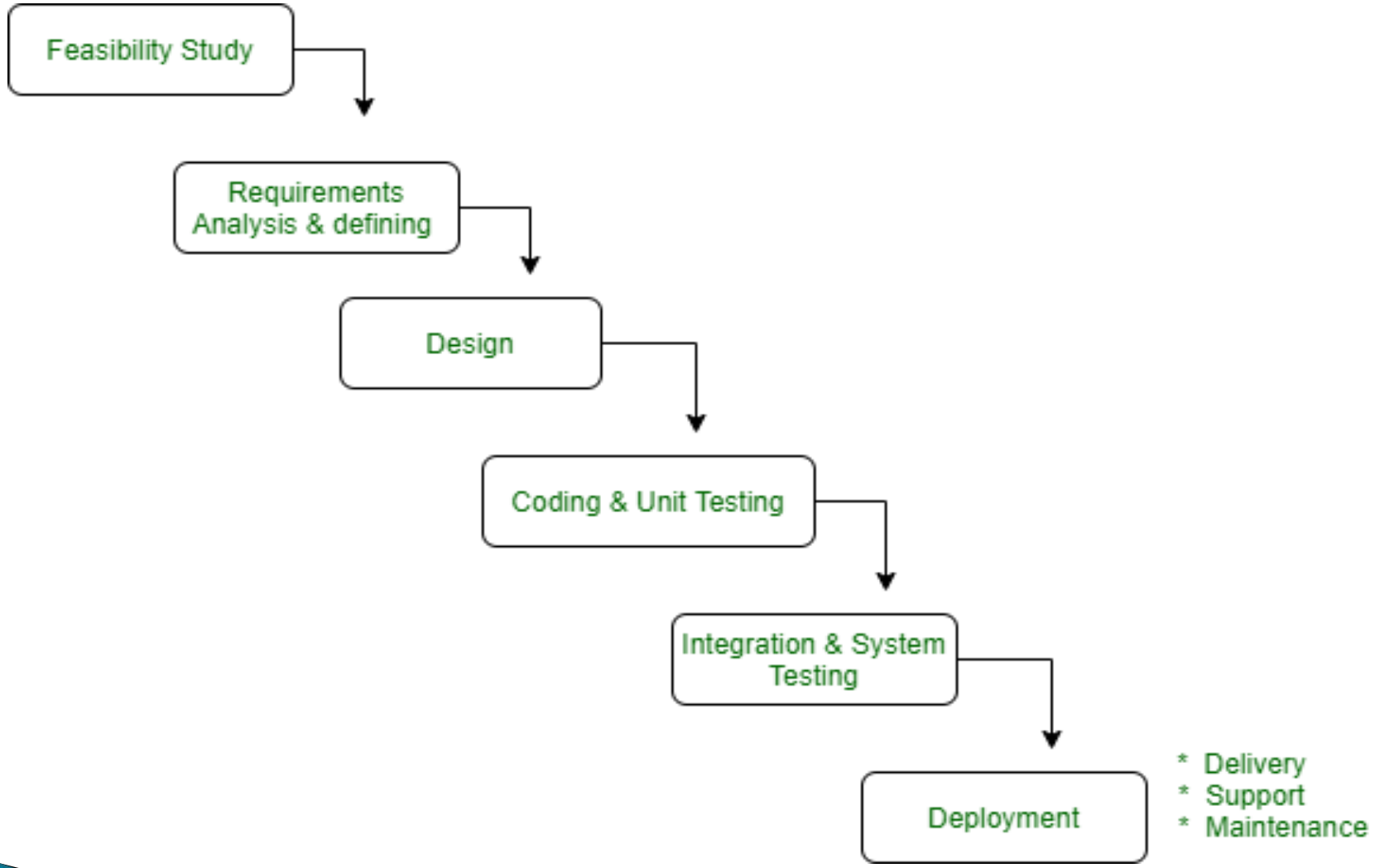
- it is not a practical model in the sense that it can not be used in actual software development projects. Thus, this model can be considered to be a theoretical way of developing software. But all other life cycle models are essentially derived from the classical waterfall model. So, in order to learn other life cycle models it is necessary to learn the classical waterfall model.

# Continued……

- Classical waterfall model divides the life cycle into the following phases :

- Feasibility Study
- Requirements Analysis and Specification
- Design
- Coding and Unit Testing
- Integration and System Testing
- Maintenance

# Continued……


Feasibility Study
Requirements Analysis & defining
Design
Coding & Unit Testing
Integration & System Testing
Deployment
* Delivery
* Support
* Maintenance

# Activities in each phase of the life cycle

**Activities undertaken during feasibility study**

- The main aim of feasibility study is to determine whether it would be **financially and technically** feasible to develop the product.
- At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting client side.
- They study different input data and output data and processing require on these data.

# Continued…..

- After overall understanding of the problem they investigate the different solutions that are possible.
- Then Examine each solution in terms of Resources, Cost and Time.
- Based on this analysis they pick the best solution and determine whether the solution is feasible financially and technically.
- They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expert in the area of development.

# Activities undertaken during Requirement analysis and specification

- The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly.

- This phase consists of two distinct activities, namely
  - ✓ Requirements gathering and analysis, and
  - ✓ Requirements specification

# Continued…..

- **The goal of the requirements gathering activity** is to collect all relevant information from the customer regarding the product and clearly understand the customer requirements so that incompleteness and inconsistencies are removed.

- **The requirements analysis activity** is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions.

# Continued…..

- For example, to perform the requirements analysis of a business accounting software required by an organization, the analyst might interview all the accountants of the organization to know their requirements.

- The data collected from such a group of users usually contain several contradictions and ambiguities, since each user typically has only a partial and incomplete view of the system.

# Continued…..

- Therefore it is necessary to identify all ambiguities and contradictions in the requirements and resolve them through further discussions with the customer.

- After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start.

- During this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document.

# Activities undertaken during design

- The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.
- During the design phase the software architecture is derived from the SRS document.

# Continued…..

- Two different approaches are available for design :

- **Traditional design approach**

  In this first structure analysis is performed where the structure of the problem is examined and than structure design is performed.

- **Object-oriented design approach**

  In this first Different object identified and than relationship among these objects are identified and than detailed design is performed.

# Activities undertaken during coding and unit testing

- The purpose of the coding and unit testing phase (sometimes called the implementation phase) of software development is to translate the software design into source code.

- Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested.

- During this phase, each module is unit tested to determine the correct working of all the individual modules.

# Activities undertaken during integration and system testing

- During the integration and system testing phase, the modules are integrated in a planned manner.
- The different modules are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps.
- During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it.
- Finally, when all the modules have been successfully integrated and tested, system testing is carried out.

# Continued…..

- The goal of system testing is to ensure that the developed system conforms to its requirements laid out 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.

# Continued…..

- System testing is normally carried out in a planned manner according to the system test plan document.
- The system test plan identifies all testing related activities that must be performed, specifies the schedule of testing, and allocates resources.
- It also lists all the test cases and the expected outputs for each test case.

# Activities undertaken during maintenance

- Maintenance of a typical software product requires much more than the effort necessary to develop the product itself.
- Many studies carried out in the past confirm this and indicate that the relative effort of development of a typical software product to its maintenance effort is roughly in the40:60 ratio.
- Maintenance involves performing any one or more of the following three kinds of activities :

# Continued…..

- Correcting errors that were not discovered during the product development phase. This is called **corrective maintenance.**

- Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called **perfective maintenance.**

- Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called **adaptive maintenance**.

# Shortcomings of the classical waterfall model

- No error checking or backtracking at the end of the life cycle phase.

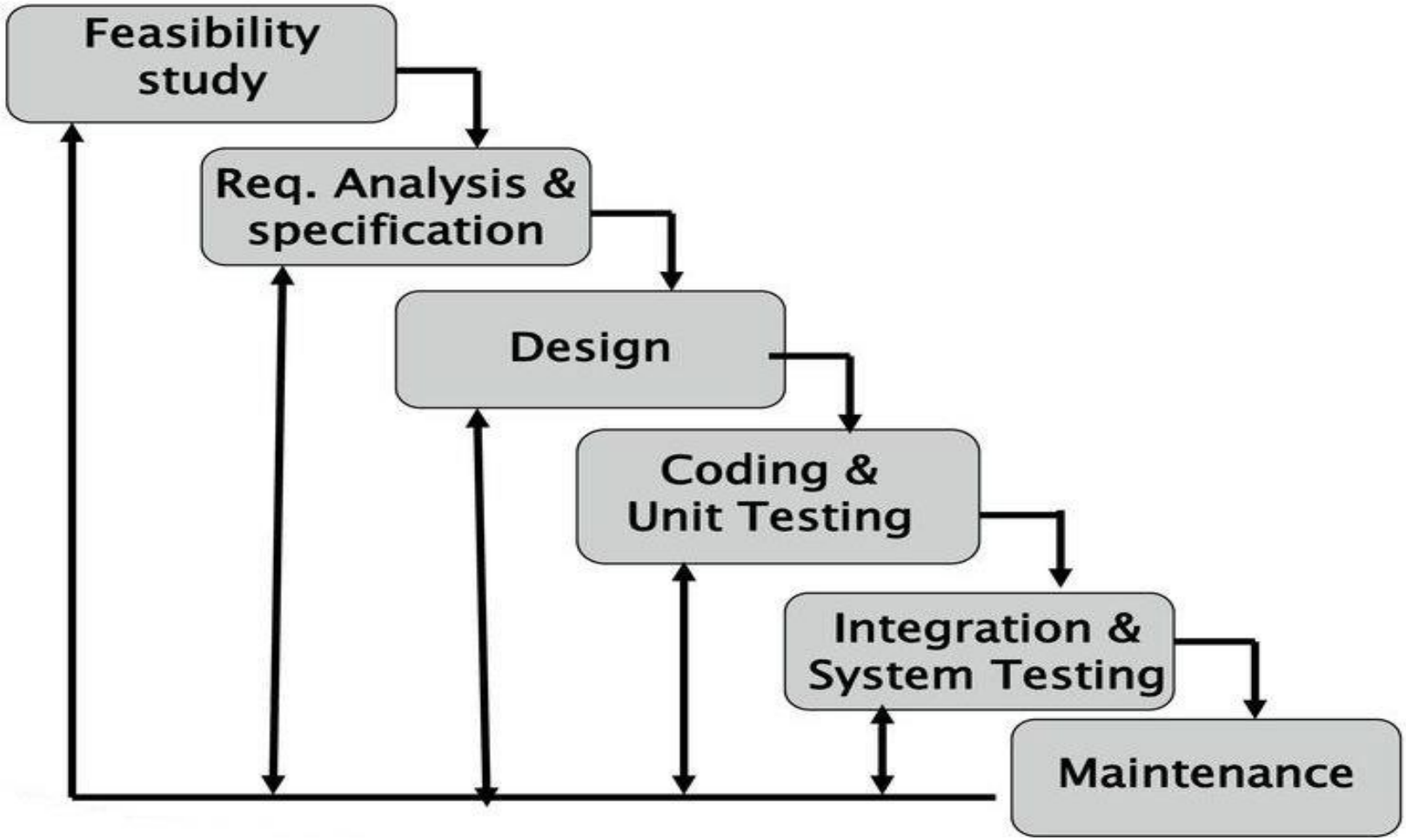
- However, in practical development environments, there are large number of errors in almost every phase of the life cycle

  (wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc.)

- These defects usually get detected much later in the Waterfall life cycle model. For example, a design defect might go unnoticed till we reach the coding or testing phase. so now correct this defect  is very difficult.

- Therefore, in any practical software development work, it is not possible to strictly follow the classical waterfall model.

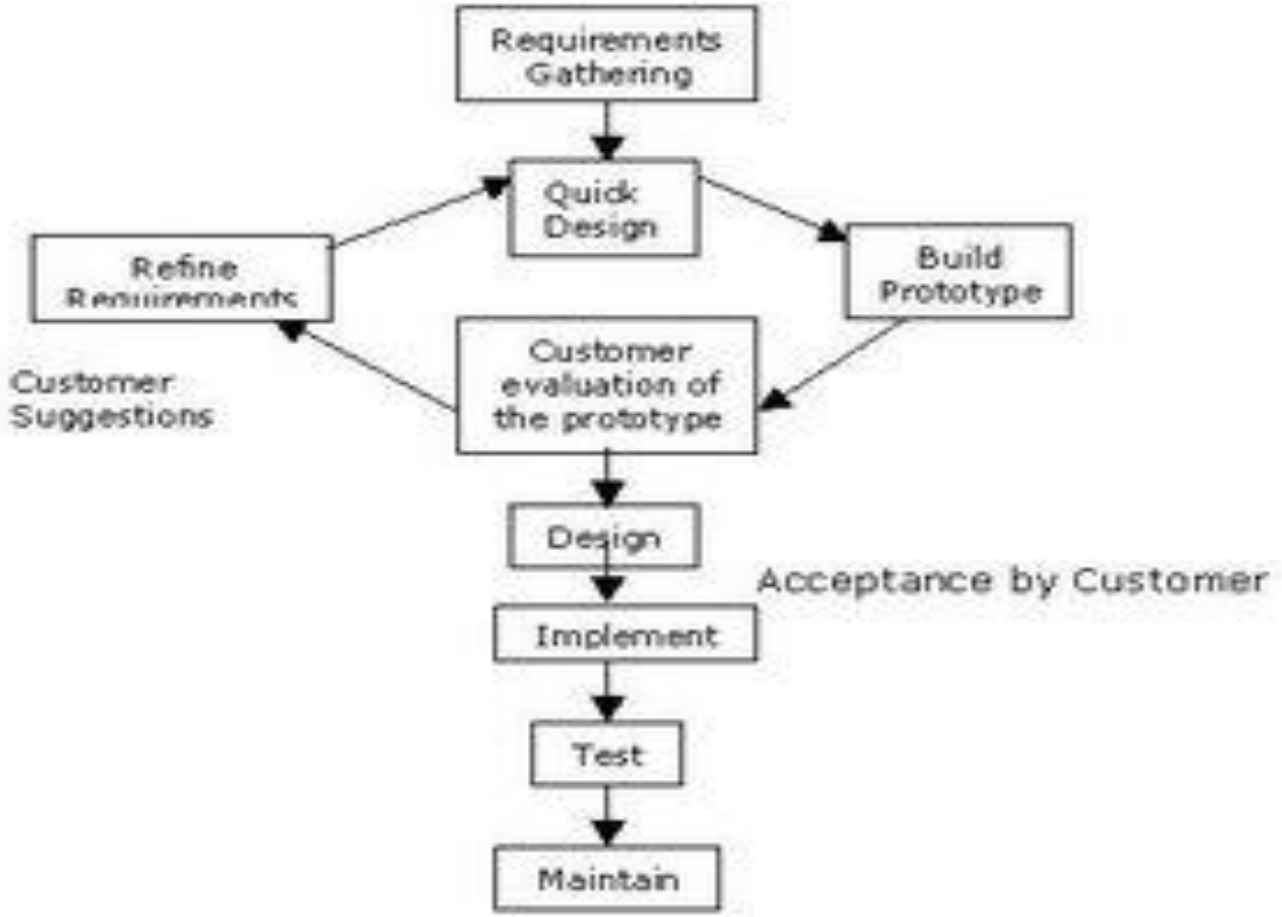
# Iterative waterfall model



# Continued…

- Feedback paths are added in classical waterfall model as shown in the figure.
- Classical waterfall model with this feedback path is known as a iterative waterfall model.
- In this model Correction of errors is done during the phase in which they occur.
- So in this model it is very easy to handle the errors compare to classical model.
- The principal of detecting error as close to their points of introduction as possible is known as a PHASE CONTAINMENT OF ERRORS.
- This is very important principal of SE.

# Prototyping Model



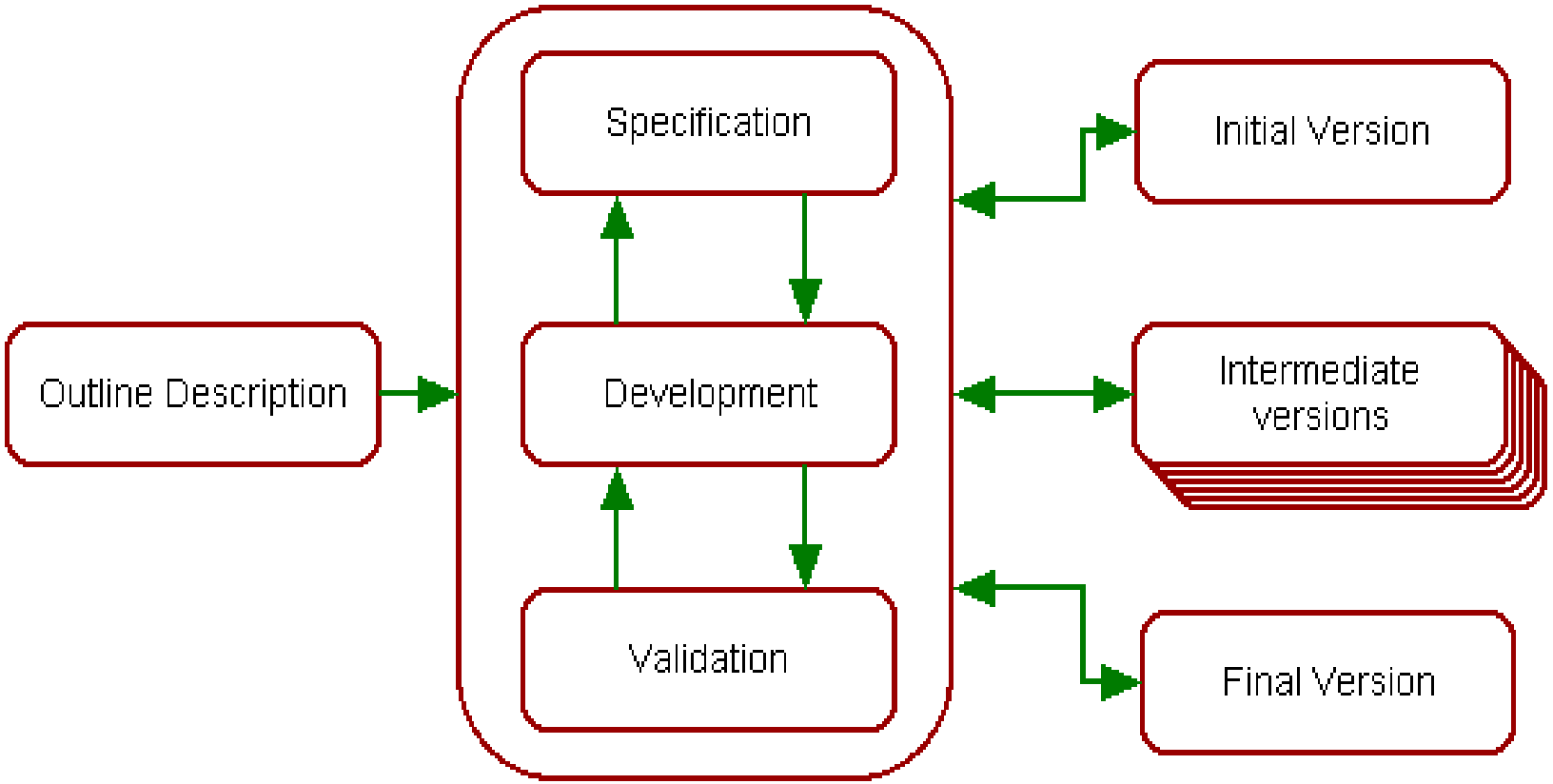
# Continued…

- Before actual software build a prototype of the system.
- Prototype is a toy implementation of the system.
- Prototype has limited functionalities, low reliability and inefficient performance.
- This model is used when user first want to small working model and than actual software or system.
- It is also used when technical solutions are not clear to the development team. In this case developer can make prototype and than using this prototype they can solve the technical issues.
- When it is not possible to ‘get it right’ the first time than we can first develop prototype and than develop a software.

# Continued…

- As shown in figure in this model development starts with an initial requirements gathering phase.
- Than quick design is carried out and prototype is built and it is submitted to customer for evaluation.
- Based on customer feedback requirements are redefined and the prototype is modified.
- This cycle continues till the customer approves the prototype.
- Than actual system is developed using iterative waterfall model.
- By making prototype and submitting it for user evaluation , many customer requirements get properly defined and technical issues get resolved.
- So change requests from customer is minimum and redesign cost also minimum.
- So over all development cost is less compare to iterative waterfall model.

# Evolutionary model



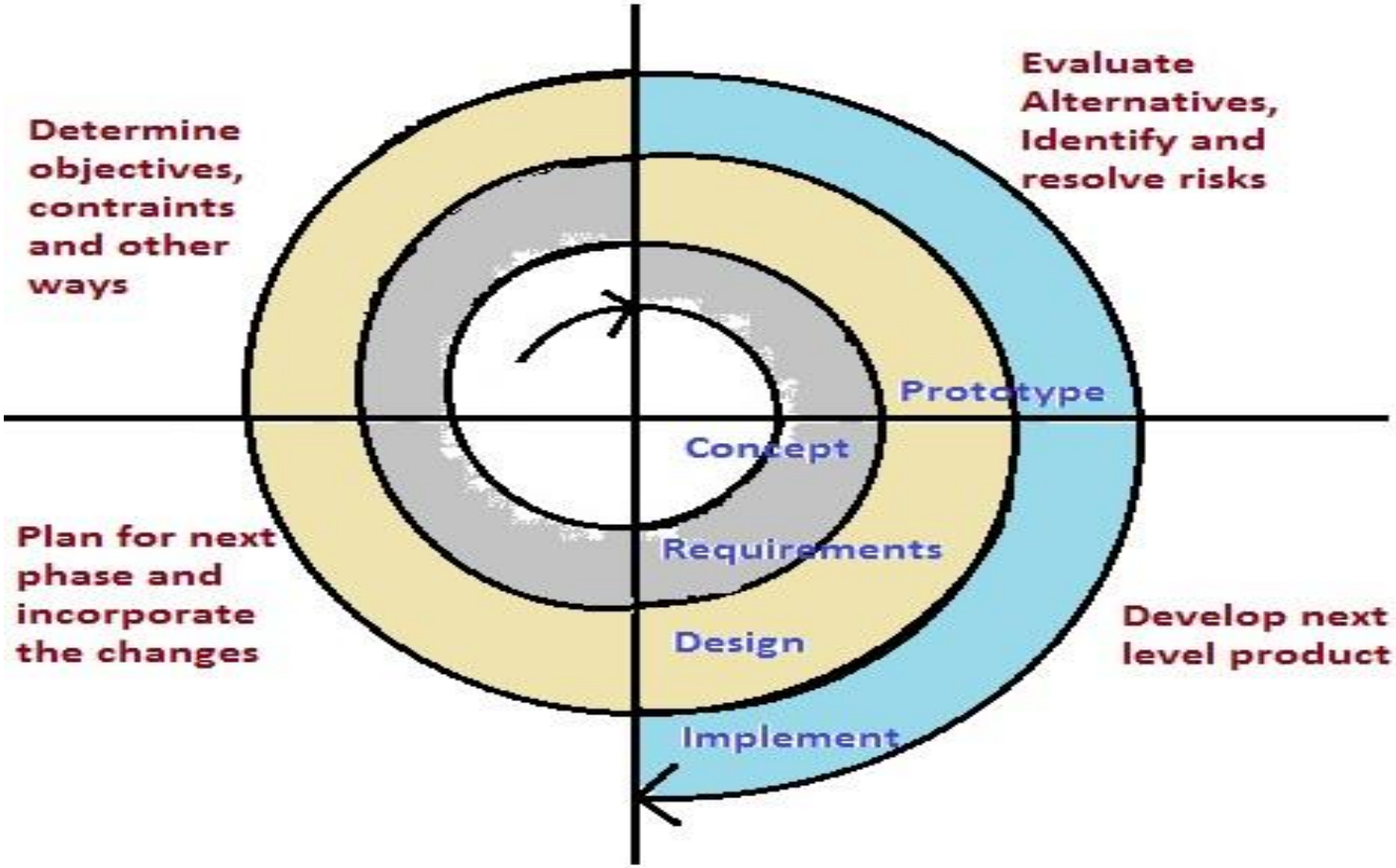
# Continued…

- It is also known as a successive versions model or incremental model.
- In this model first module is broken down into several modules which can be incrementally constructed and delivered.
- First core module is developed and than new functionalities added in successive versions. each version may be developed using iterative waterfall model.
- Each successive versions is more useful than previous versions.
- In this model user can get chance to work with partially develop software before complete system.

# Continued…

- After delivery of software changes are minimum in this model.
- Core module is tested thoroughly so chances of errors are very less.
- In this model no need of large resources at a time because system is developed in module.
- The main disadvantage of this model is that it is very difficult to divide the problem in several units and than incrementally implemented and delivered.
- So it is for only very large products and it is also used in Object oriented software projects where system can easily divide in terms of objects.
- If customer prefers to receive product in one by one module rather than full product than only this module is used.

# Spiral Model



# Continued…

- Fig. shows that spiral model contain many loops and each loop of the spiral represents a phase of the software process.
- For example innermost loop for feasibility study the next loop for requirement analysis and next for design and so on.
- Each phase in this model is divide into four sectors.
- The first sector identifies the objectives of the phase and alternative solutions.
- During second part alternative solution are evaluated to select the best solution possible.
- For chosen solution the **risks** are identified and dealt with that by developing an appropriate prototype.(Risk is any unwanted event that might hamper the successful completion of a software project)

# Continued…

- Activities during third part is developing and verifying the next level of the product or software.
- Activities during fourth part is review the result of activities done so far with customer and planning the next iteration of the spiral.
- After some iterations the risks are resolved and the software is ready for development.
- Than using iterative waterfall model the software development is done.
- Radius of the spiral indicate cost of the project and angular dimension represents the process made in the current phase.
- Risk handling is most important features of this model.

# Continued…

- So the spiral model can be viewed as a **meta model** because it uses the features of all model.
- In spiral model uses a prototyping model as a risk reduction before actual development.
- It also supports the evolutionary model because iterations along spiral can ne considered as a one level of evolutionary model.
- After risk reduction by prototype it uses the stepwise approach of the water fall model.

# Comparison of different life cycle models.

- The classical waterfall model can be considered as the basic model of all model. but it supports no mechanism to handle the errors during any phase so can not be used in practical development projects.
- This problem is overcome in iterative waterfall model because feedback path is added in each phase.
- It is very simple to understand and use so most widely used in software development.
- This model is used only for well understood problems(all requirement is clear and technical issues also clear).
- It is not used for very large projects and for the projects with many risks.

# Continued…

- The prototyping model is used when either user requirements are not clear or technical issues are not clear.
- The evolutionary model is suitable for large problems which can be divided in to several modules.
- It is mainly used in object oriented development projects.
- It is only used if the customer accept the incremental delivery of software(one by one module)
- Spiral model is a meta model used features of all other model.
- It is mostly used in project having many risks.
- It is very complex model than other so generally it is not used in ordinary projects.

# Agile Manifesto

- **Individuals and interactions** − In Agile development, self-organization and motivation are important.
- **Working software** − Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
- **Customer collaboration** − As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
- **Responding to change** − Agile Development is focused on quick responses to change and continuous development.

# Agility Principles (12 Principal of Agile Software)

- Agile promotes early and continuous delivery of valuable software.
- Being flexible about change at any point of development.
- Working on frequent and short deliveries throughout the development process.
- Agile promotes transparency between business people and developers and requires them to work together
- Agile advocates for motivating individuals in the team.
- Agile believes in face to face communication as the most effective way to communicate.

# Continue…

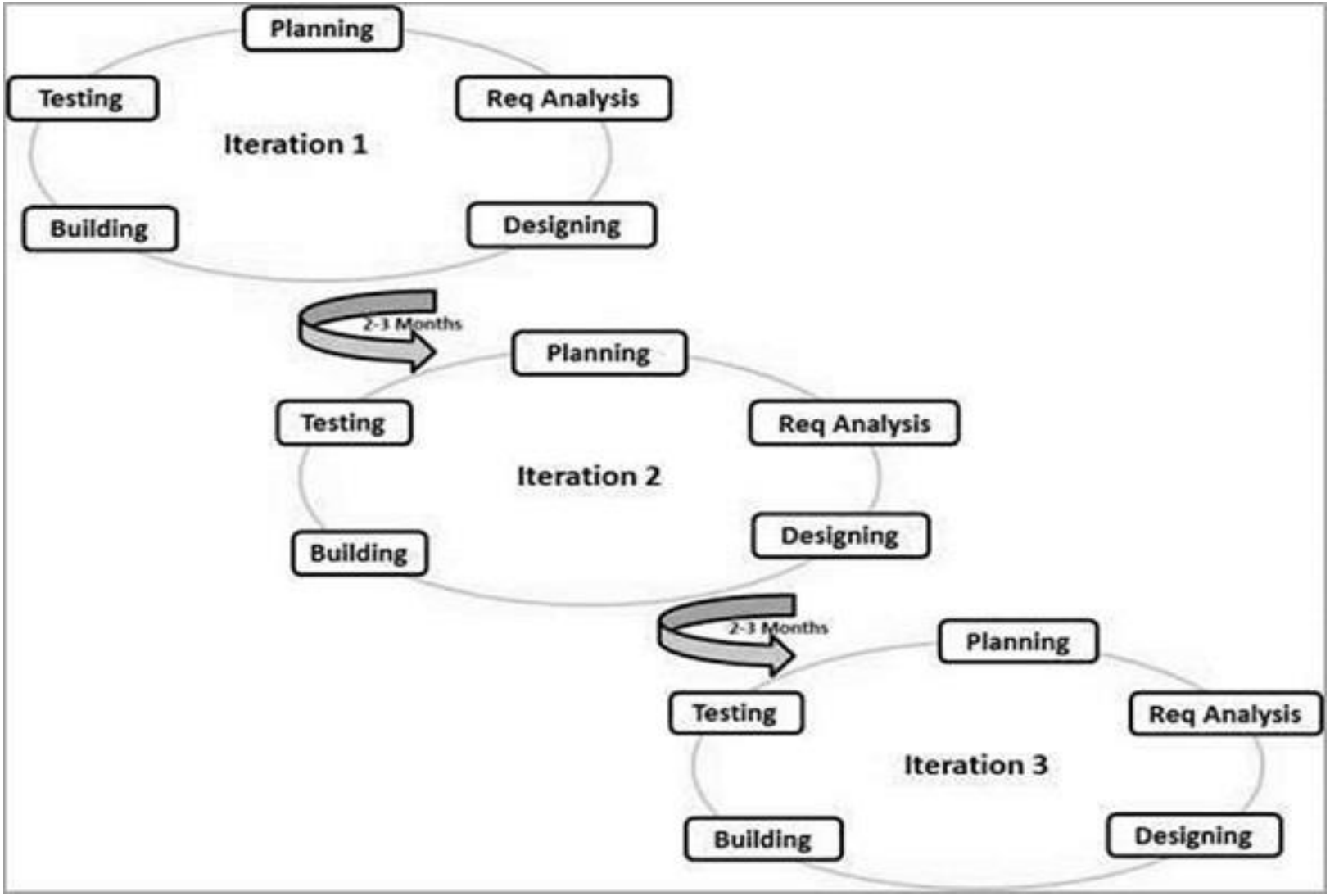
- Agile motivates continuous attention towards effective designing and technical excellence through following optimal code standard.
- Agile promotes sustainable development.
- Working software is the primary measure of progress.
- Agile promotes removing unnecessary tasks and prioritizing activities can have maximum impact with less and effective work done.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

# Agile Development Model

- Agile refers to something that is quick or adaptable. A software development approach based on iterative development is referred to as an “agile process model.”
- Agile approaches divide projects into smaller iterations or sections and avoid long-term planning.
- The scope and requirements of the project are defined at the start of the development phase.
- The number of iterations, duration, and scope of each iteration are all clearly determined ahead of time.
- Unlike the Waterfall model, both development and testing operations are perform at the same time under the Agile model of software testing.

# How it Works?

- Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements.
- In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.
- Iterative approach is taken and working software build is delivered after each iteration.
- Each build is incremental in terms of features; the final build holds all the features required by the customer.


Planning
Testing
Req Analysis
Iteration 1
Building
Designing
2-3 Months
Planning
Testing
Req Analysis
Iteration 2
Building
Designing
2-3 Months
Planning
Testing
Req Analysis
Iteration 3
Building
Designing

# Agile Vs Traditional SDLC Models

- Agile is based on the **adaptive software development methods**, whereas the traditional SDLC models like the waterfall model is based on a predictive approach. Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle.
- Predictive methods entirely depend on the **requirement analysis and planning** done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

# Continue…

- Agile uses an **adaptive approach** where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.
- **Customer Interaction** is the backbone of this Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

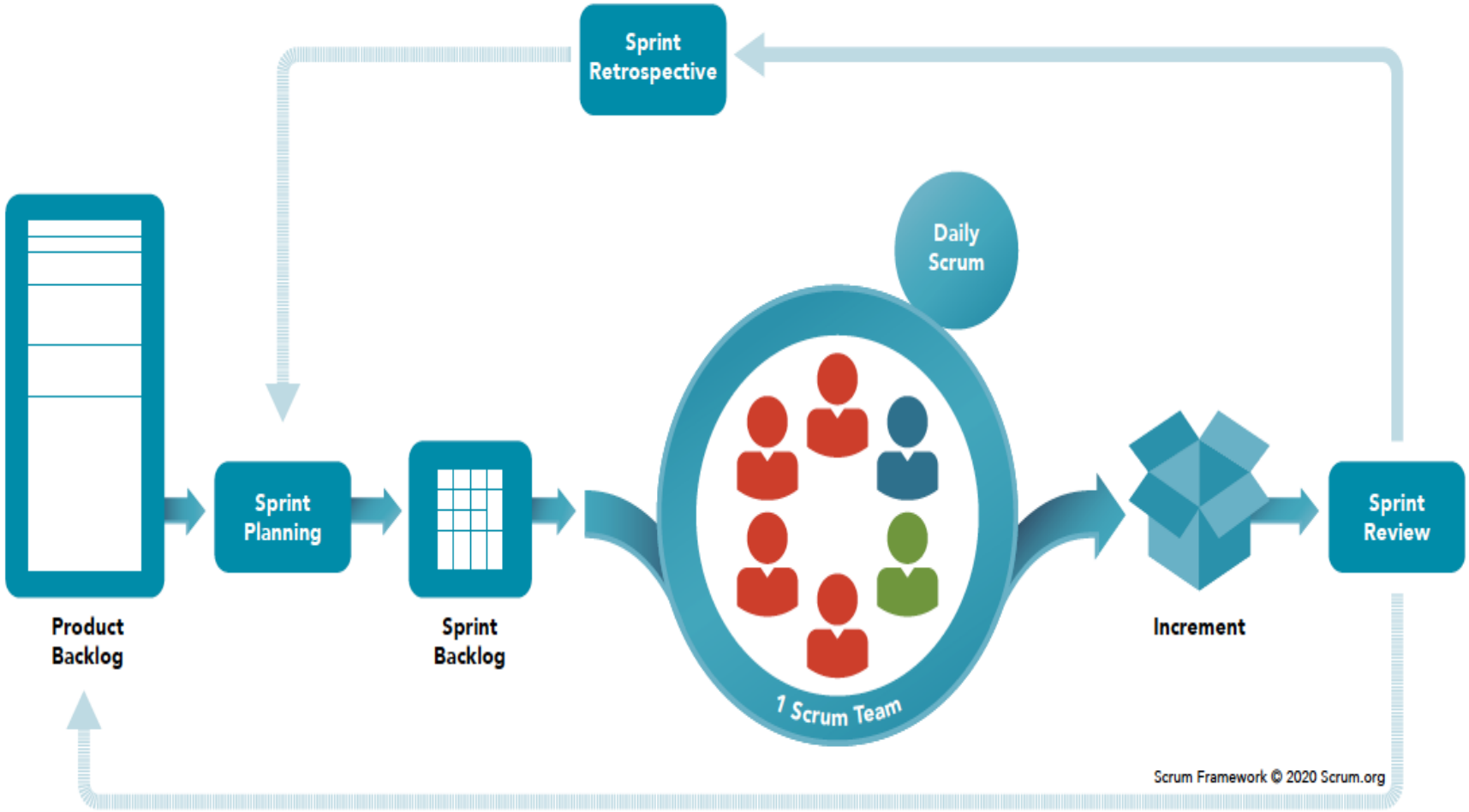
# Types of widely used Agile Models

- **Scrum**
- Crystal
- Dynamic Software Development Method(DSDM)
- Feature Driven Development(FDD)
- Lean Software Development
- **extreme Programming(XP)**
- Kanban

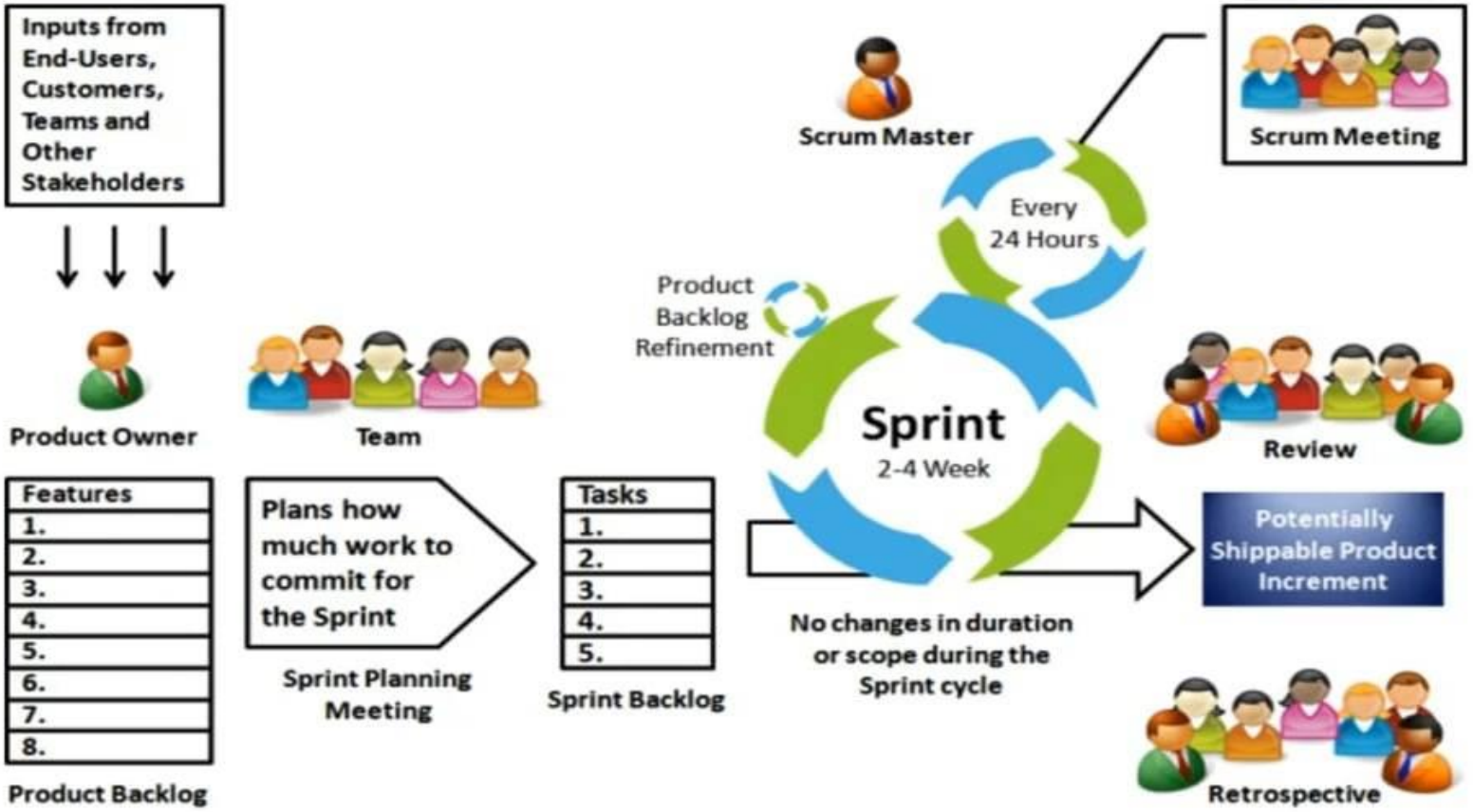
# Scrum

- Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems.
- powerful set of values, principles, and practices.
- Scrum relies on cross-functional teams to deliver products and services in *short cycles* enabling:
- Fast feedback
- Quicker innovation
- Continuous improvement
- Rapid adaptation to change
- More delighted customers
- Accelerated pace from idea to delivery

# Scrum Framework



# SCRUM


Inputs from End-Users, Customers, Teams and Other Stakeholders
Product Owner
Team
Features
1.
2.
3.
4.
5.
6.
7.
8.
Product Backlog
Plans how much work to commit for the Sprint
Sprint Planning Meeting
Tasks
1.
2.
3.
4.
5.
Sprint Backlog
Scrum Master
Every 24 Hours
Product Backlog Refinement
Sprint
2-4 Week
No changes in duration or scope during the Sprint cycle
Scrum Meeting
Review
Potentially Shippable Product Increment
Retrospective

# The Scrum Team

- A scrum has three accountabilities (commonly known as roles):
- **Developers** – On a scrum team, a developer is anyone on the team that is delivering work.
- **Product Owner** – Holds the vision for the product and prioritizes the product backlog
- **Scrum Master** – Helps the team best use scrum to build the product.
- The scrum team works together to achieve a shared goal and deliver value to users of their product or service.

# Scrum Events

- Scrum teams work in sprints, each of which includes several events (or activities). Don't think of these events as meetings or ceremonies; the events that are contained within each sprint are valuable opportunities to inspect and adapt the product or the process (and sometimes both).
- **The Sprint** – The heartbeat of scrum. Each sprint should bring the product closer to the product goal and is a month or less in length.
- **Sprint Planning** – The entire scrum team establishes the sprint goal, what can be done, and how the chosen work will be completed. Planning should be timeboxed to a maximum of 8 hours for a month-long sprint, with a shorter timebox for shorter sprints.

# Continue…

- **Daily Scrum** – The developers (team members delivering the work) inspect the progress toward the sprint goal and adapt the sprint backlog as necessary, adjusting the upcoming planned work. A daily scrum should be time boxed to 15 minutes each day.
- **Sprint Review** – The entire scrum team inspects the sprint's outcome with stakeholders and determines future adaptations. Stakeholders are invited to provide feedback on the increment.
- **Sprint Retrospective** – The scrum team inspects how the last sprint went regarding individuals, interactions, processes, tools, and definition of done. The team identifies improvements to make the next sprint more effective and enjoyable. This is the conclusion of the sprint.
- sprint reviews are about the **product**; sprint retrospectives are about the **process** of creating that product.

# Scrum Artifacts

- Scrum artifacts help manage the work

- **Product Backlog** – An emergent, ordered list of what is needed to improve the product and includes the product goal.
- **Sprint Backlog** – The set of product backlog items selected for the sprint by the developers (team members), plus a plan for delivering the increment and realizing the sprint goal.
- **Increment** – A sum of usable sprint backlog items completed by the developers in the sprint that meets the definition of done, plus the value of all the increments that came before. Each increment is a recognizable, visibly improved, operating version of the product.
- The team displays its plans and progress so that all team members and stakeholders can always see what the team is accomplishing.

# How It All Works Together

- Scrum requires a Scrum Master to foster an environment where:

1) A Product Owner orders the work for a complex problem into a Product Backlog.
2) The Scrum Team turns a selection of the work into an Increment of value during a Sprint (Sprint planning & Backlog) .
3) Daily scrum to check progress
4) At the end of sprint review it and deliver.
5) Plan for the next sprint
6) *Repeat*

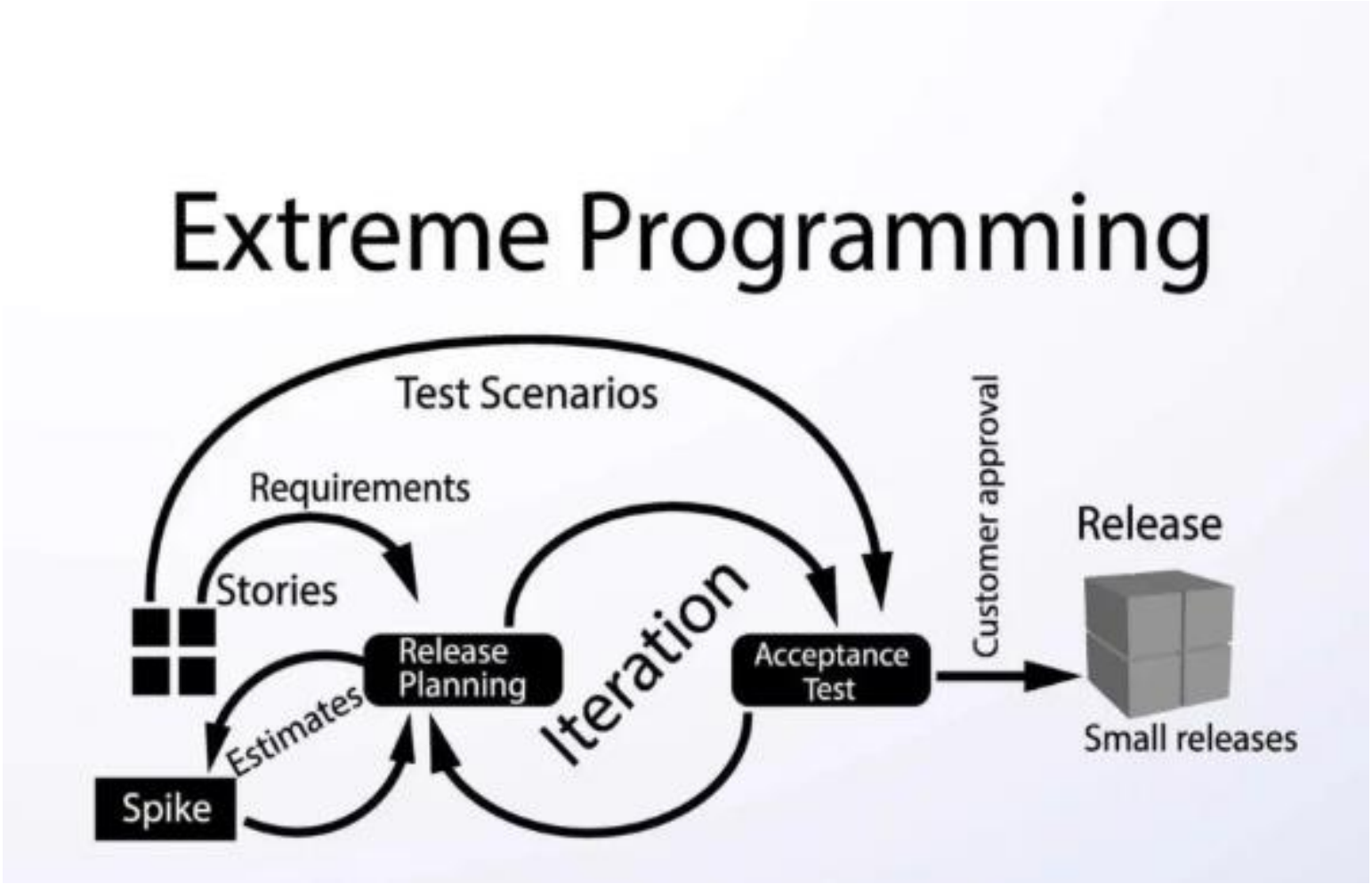
# Extreme Programming(XP)

- XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.
- Extreme programming is a software development methodology that's part of what's collectively known as agile methodologies.
- XP is built upon values, principles, and practices, and its goal is to allow small to mid-sized teams to produce high-quality software and adapt to evolving and changing requirements.
- The team is expected to self-organize.

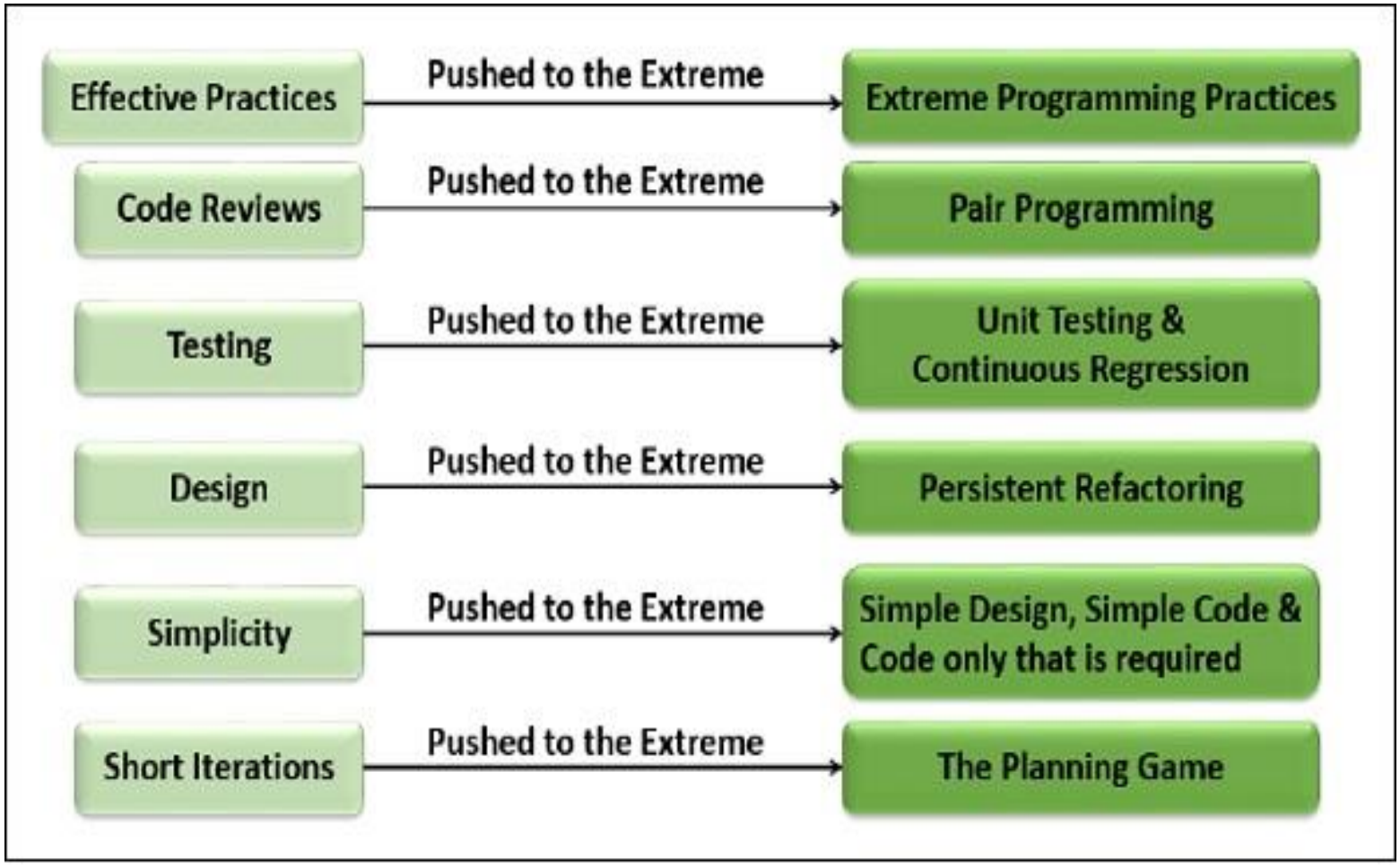
# Continue…

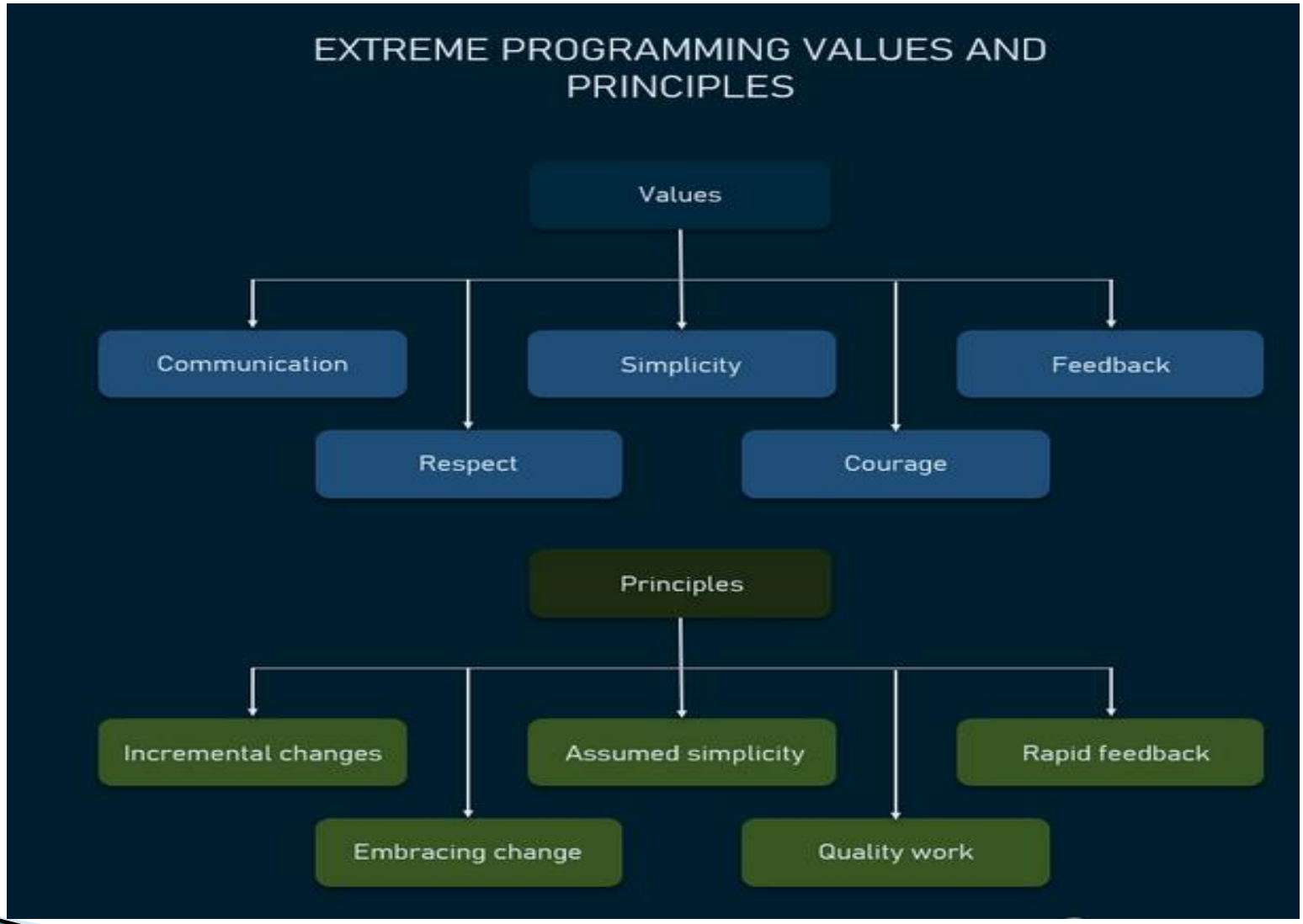
- What sets XP apart from the other agile methodologies is that XP emphasizes the technical aspects of software development. Extreme programming is precise about how engineers work since following engineering practices allows teams to deliver high-quality code at a sustainable pace.
- Extreme programming is, in a nutshell, about good practices taken to an extreme. Since pair-programming is good, let's do it all of the time. Since testing early is good, let's test before the production code is even written.

# Continue…


Extreme Programming
Test Scenarios
Requirements
Stories
Release Planning
Iteration
Acceptance Test
Customer approval
Release
Small releases
Estimates
Spike

# Continue…


Effective Practices
Pushed to the Extreme
Extreme Programming Practices
Code Reviews
Pushed to the Extreme
Pair Programming
Testing
Pushed to the Extreme
Unit Testing &
Continuous Regression
Design
Pushed to the Extreme
Persistent Refactoring
Simplicity
Pushed to the Extreme
Simple Design, Simple Code &
Code only that is required
Short Iterations
Pushed to the Extreme
The Planning Game


EXTREME PROGRAMMING VALUES AND PRINCIPLES
Values
Communication
Simplicity
Feedback
Respect
Courage
Principles
Incremental changes
Assumed simplicity
Rapid feedback
Embracing change
Quality work

*Happy Learning*