– Umbrella Activities and Software Processes Models

The Software Process

- ✧ A software process is a set of related activities (actions and tasks) that leads to the production of a software system.
- ✧ Many different software processes but all involve:
  - Specification – defining what the system should do;
  - Design and implementation – defining the organization of the system and implementing the system;
  - Validation – checking that it does what the customer wants;
  - Evolution – changing the system in response to changing customer needs.
- ✧ A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Software Process Descriptions

- ✧ When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
- ✧ Process descriptions may also include:
  - Products, which are the outcomes of a process activity; For example, the outcome of the activity of architectural design may be a model of the software architecture.
  - Roles, which reflect the responsibilities of the people involved in the process; Examples of roles are project manager, configuration manager, and programmer.
  - Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced. For example, before architectural design begins, a precondition may be that the consumer has approved all requirements; and a post condition might be that the UML models describing the architecture have been reviewed.

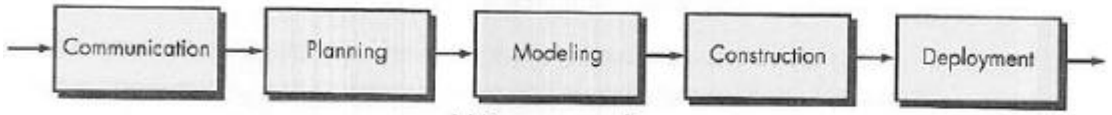
Plan-driven and Agile process

- ✧ Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- ✧ In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- ✧ In practice, most practical process include elements of both plan-driven and agile approaches.
- ✧ There are no right or wrong software process.

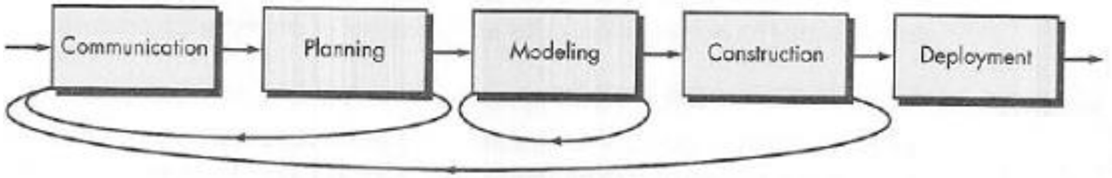
Process Flow

- *Linear process* flow executes each of the five activities in sequence.
- An *iterative process* flow repeats one or more of the activities before proceeding to the next.
- An *evolutionary process* flow executes the activities in a circular manner. Each circuit leads to a more complete version of the software.
- A *parallel process* flow executes one or more activities in parallel with other activities (modeling for one aspect of the software in parallel with construction of another aspect of the software).

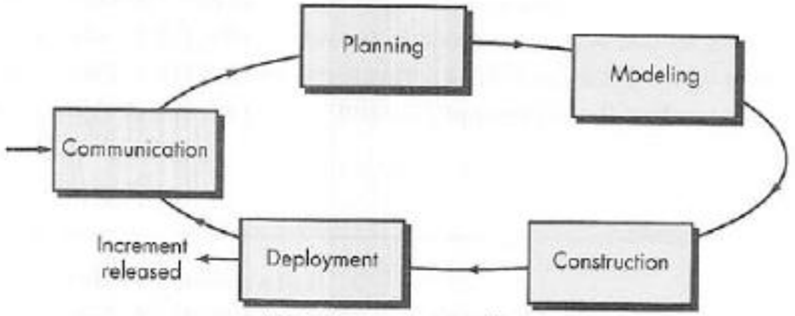
Process Flow



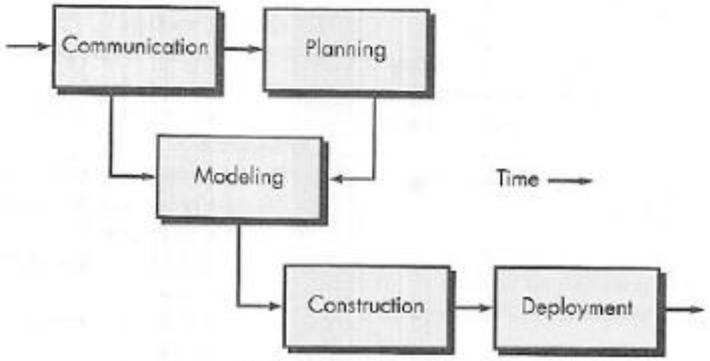
(a) Linear process flow



(b) Iterative process flow

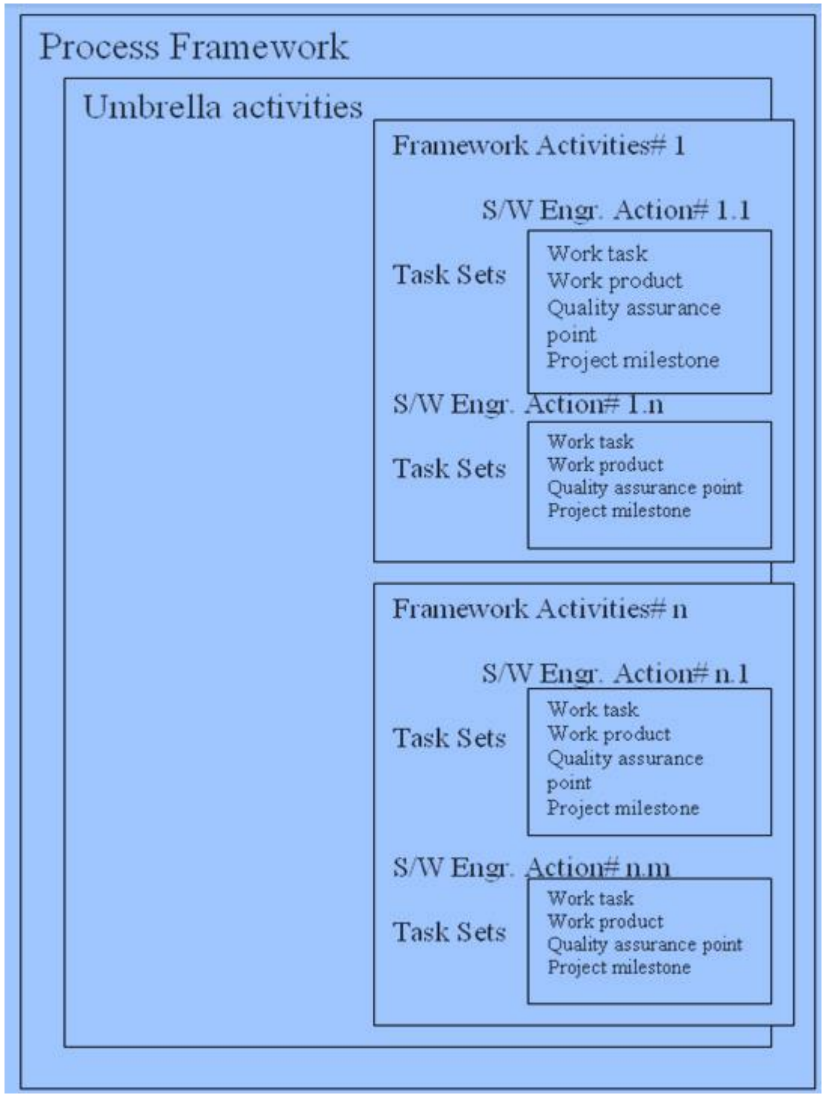


(c) Evolutionary process flow



(d) Parallel process flow

A Generic Process Model



A Generic Process Model

✧ As we discussed before, a generic process framework for software engineering defines five framework activities-communication, planning, modeling, construction, and deployment.

✧ In addition, a set of umbrella activities- project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.

✧ A set of framework activities, which are always applicable, regardless of the project type, and a set of umbrella activities, which are the non SDLC activities that span across the entire software development life cycle.

Umbrella activities include:

01. Software project tracking and control :

✧ Tracking and Control is the dual process of detecting when a project is drifting off-plan, and taking corrective action to bring the project back on track. But a successful project manager will also be able to tell when the plan itself is faulty, and even re-plan the project and its goals if necessary.

02. Formal technical reviews :

✧ This includes reviewing the techniques that has been used in the project.

Umbrella activities include:

03. <u>Software quality assurance :</u>

- ✧ This is very important to ensure the quality measurement of each part to ensure them.

04. <u>Software configuration management :</u>

- ✧ In software engineering, software configuration management (SCM or S/W CM) is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management. SCM practices include revision control and the establishment of baselines.

05. <u>Document preparation and production :</u>

- ✧ All the project planning and other activities should be hardly copied and the production get started here.

Umbrella activities include:

06. Reusability management

This includes the backing up of each part of the software project they can be corrected or any kind of support can be given to them later to update or upgrade the software at user/time demand.

07. Measurement

This will include all the measurement of every aspects of the software project.

08. Risk management :

Risk management is a series of steps that help a software team to understand and manage uncertainty. It’s a really good idea to identify it, assess its probability of occurrence, estimate its impact, and establish a contingency plan that─ ‘should the problem actually occur’.

Software Process Model

- ✧ Attempt to organize the software life cycle by
  - defining activities involved in software production
  - order of activities and their relationships
- ✧ Goals of a software process
  - standardization, predictability, productivity, high product quality, ability to plan time and budget requirements

Software Process Models

Software Process Models

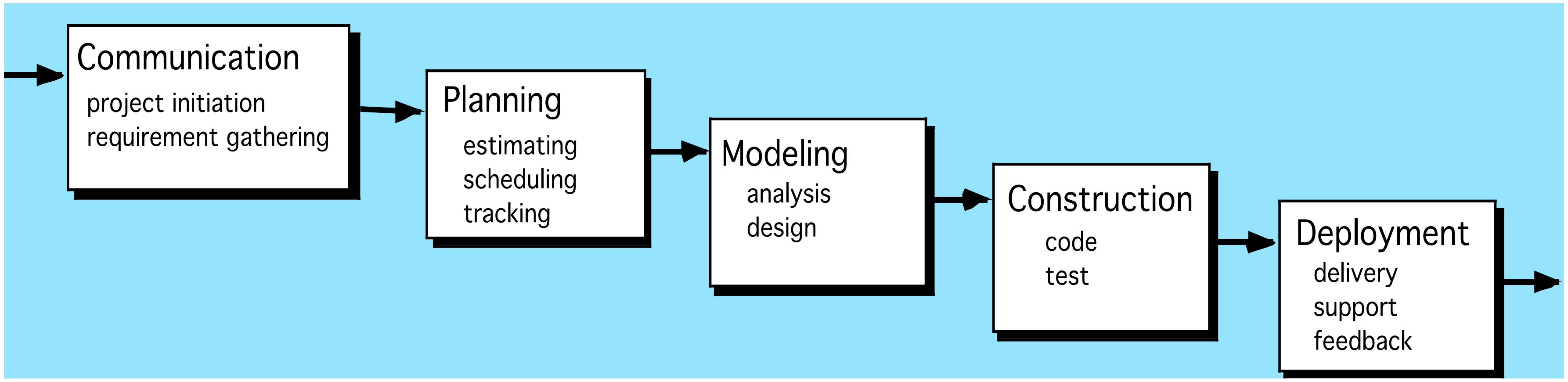
- ✧ The waterfall model
- ✧ V Model
- ✧ Incremental development
- ✧ Spiral Model
- ✧ Prototyping Model
- ✧ RAD
- ✧ Agile
- ✧ Scrum
- ✧ In practice, most large systems are developed using a process that incorporates elements from all of these models.

The Waterfall Model

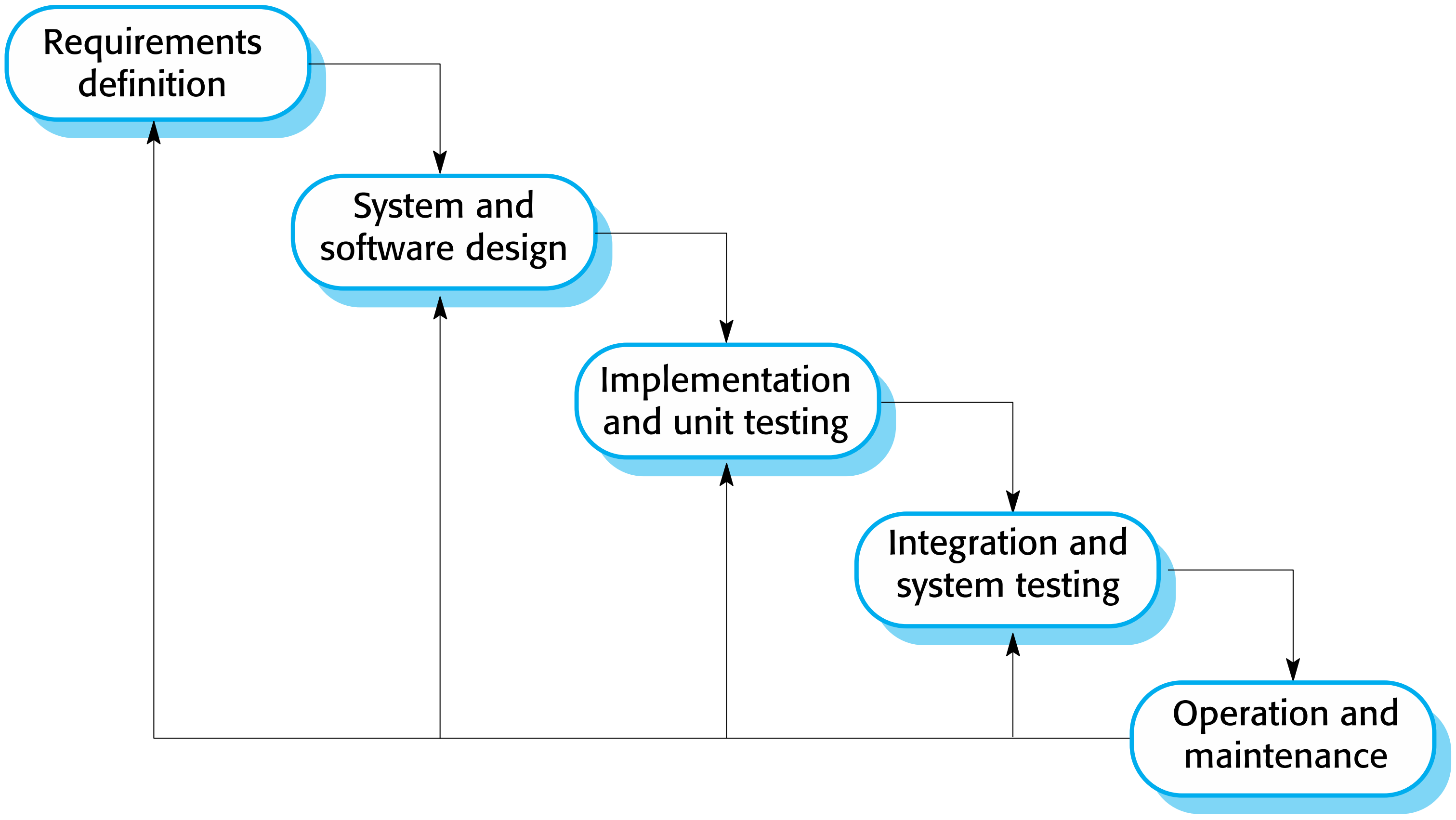
Waterfall

The Waterfall Model

- ✧ It is the oldest paradigm for SE. When requirements are well defined and reasonably stable, it leads to a linear fashion.
- ✧ The classic life cycle suggests a systematic, sequential approach to software development.



The Waterfall Model


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

Waterfall Model Phases

- ✧ The work flows from communication through deployment in a reasonably linear/sequential fashion.
- ✧ There are separate identified phases in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- ✧ In principle, a phase has to be complete before moving onto the next phase.

Advantages and Disadvantages

- **Advantages**
  - Simple and easy to understand and use.
  - Easy to manage.
  - Works well for smaller and low budget projects where requirements are very well understood.
  - Clearly defined stages and well understood.

- **Disadvantages**
  - The customer must have patience. A working version of the program(s) will not be available until late in the project time span.
  - Real projects rarely follow the sequential flow that the model proposes.
  - High amounts of risk and uncertainty.
  - The main drawback of the **waterfall model** is the difficulty of accommodating change after the process is underway.

When To Use?/Best Suited for:

- ✧ Requirements are well defined and fixed that may not change.
- ✧ There are no ambiguous requirements (no confusion).
- ✧ The project is short and cost is low.
- ✧ The software projects where Risk is zero or minimum.
- ✧ **Examples:** Mission Critical Projects, Embedded system.

V Model

Waterfall Model Variation – V Model

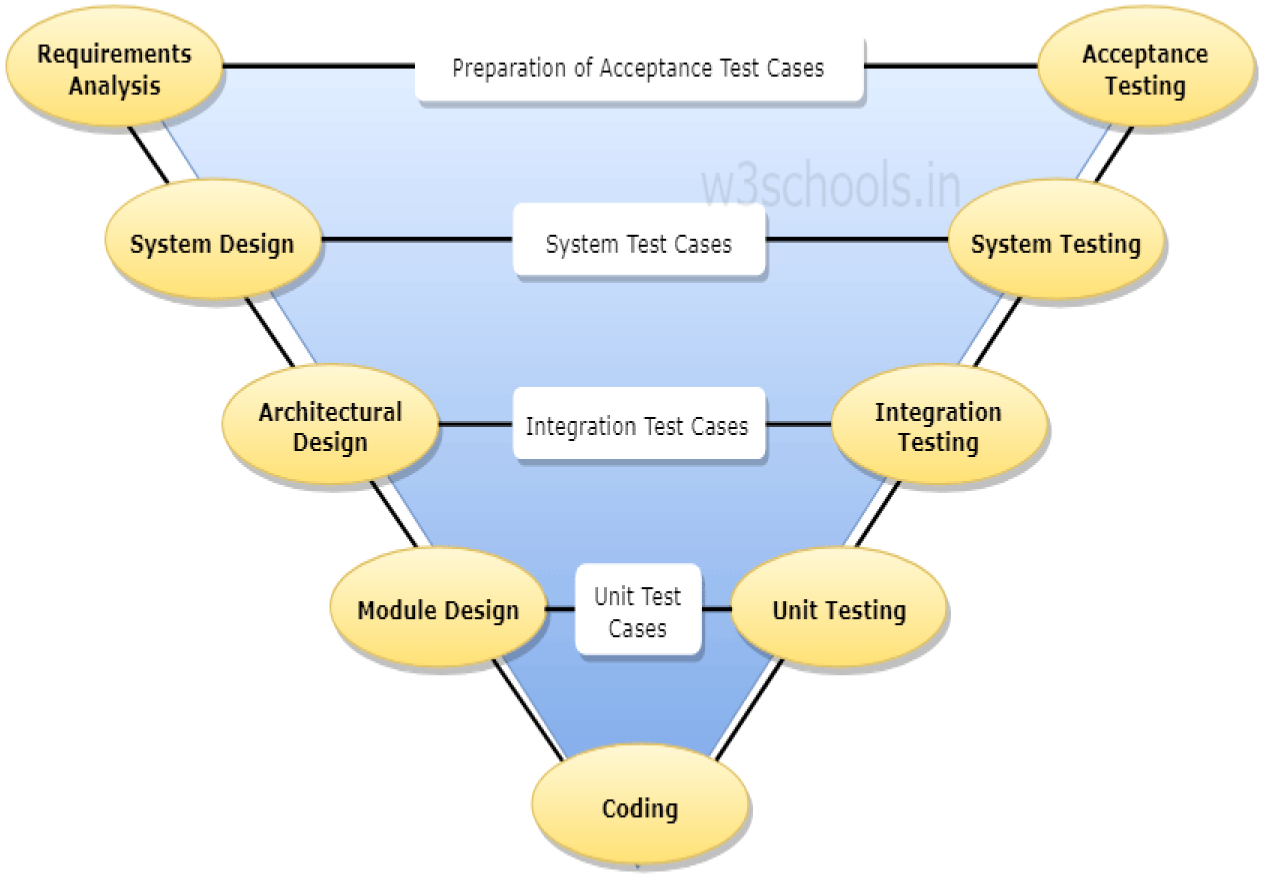


Fig: The Design of the SDLC V-Model

V Model

- ✧ A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early code construction activates.

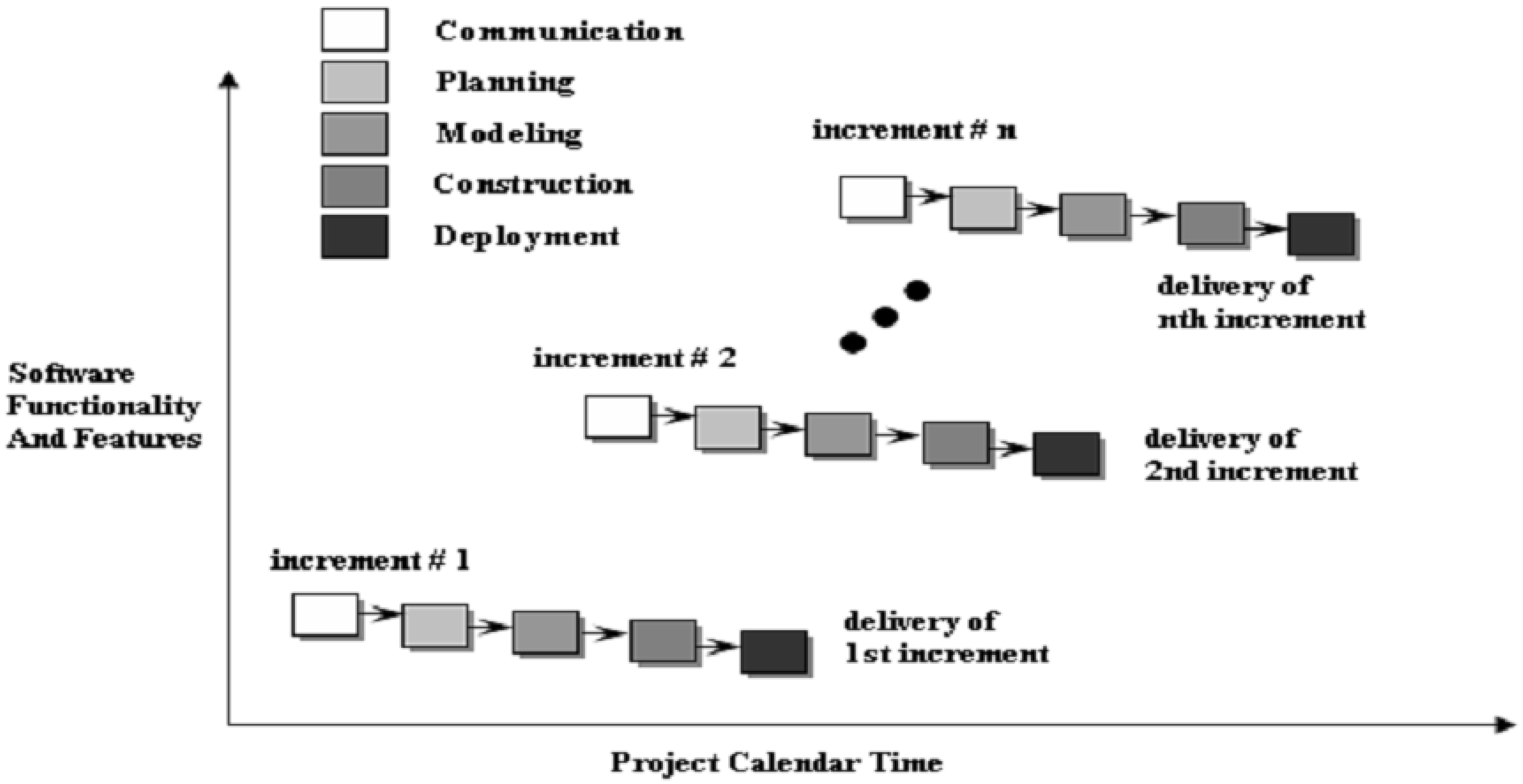
- ✧ Team first moves down the left side of the V to refine the problem requirements. Once code is generated, the team moves up the right side of the V, performing a series of tests that validate each of the models created as the team moved down the left side.

V and V Combined

- **Verification**
  - In the concept of verification in the V-Model, static analysis technique is carried out without executing the code. This evaluation procedure is carried out at the time of development to check whether specific requirements will meet or not.
- **Validation**
  - This concept of V-Model comprises of dynamic analysis practice (both functional as well as non-functional), and testing is done by code execution. The validation of a product is done once the development is complete for determining if the software meets up the customer hope needs.
- So both verification and validation are combined and work in parallel to make the V-Model fully functional.

Incremental Model

Incremental Model



Incremental Model

- Incremental development is based on the idea of developing an initial implementation, getting feedback from users and others, and evolving the software through several versions until the required system has been developed
- There are many situations in which initial software requirements are reasonably well defined but the overall scope of the development effort precludes a purely linear process.
- The **incremental model** combines elements of the waterfall model applied in an iterative fashion.
- This approach can be either plan-driven, agile or, more usually, a mixture of these approaches.
- The incremental model applies linear sequences in a staggered fashion as calendar time progresses.

Incremental Model

- Each linear sequence produces deliverable —increments of the software.
- When an incremental model is used, the first increment is often a **core product**.
- The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature.
- Each increment or version of the system incorporates some of the functionality that is needed by the customer. Generally, the early increments of the system include the most important or most urgently required functionality.

Incremental Model - Example

- Word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the **first increment**; more sophisticated editing and document production capabilities in the **second increment**; spelling and grammar checking in the **third increment**; and advanced page layout capability in the **fourth increment**.

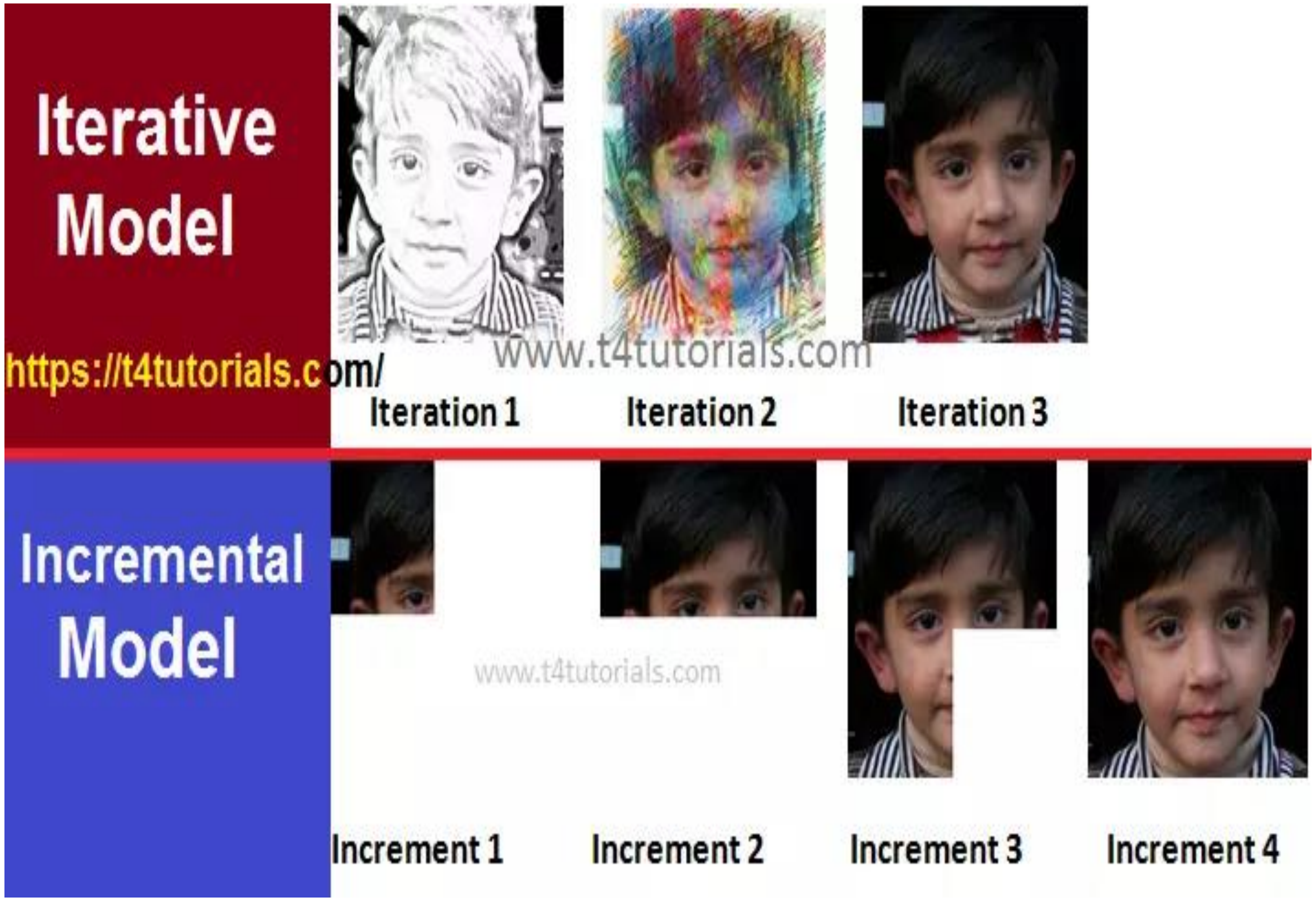
Advantages and Disadvantages

- **Advantages**
  - Cost is distributed with less human power & staffing.
  - Errors are simultaneously corrected.
  - It is easier to get customer feedback on the development work that has been done. Customers can comment on demonstrations of the software and see how much has been implemented.
  - Early delivery and deployment of useful software to the customer is possible, even if all of the functionality has not been included. Customers are able to use and gain value from the software earlier than is possible with a waterfall process.
  - Testing is easy.

Advantages and Disadvantages

- **Disadvantages**
  - Requires proper planning to distribute the work.
  - Total cost required for the development of the product is high.
  - The process is not visible. Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost effective to produce documents that reflect every version of the system.
  - System structure tends to degrade as new increments are added. Regular change leads to messy code as new functionality is added in whatever way is possible. It becomes increasingly difficult and costly to add new features to a system. To reduce structural degradation and general code messiness, agile methods suggest that you should regularly refactor (improve and restructure) the software.

Iterative VS incremental



Evolutionary Models

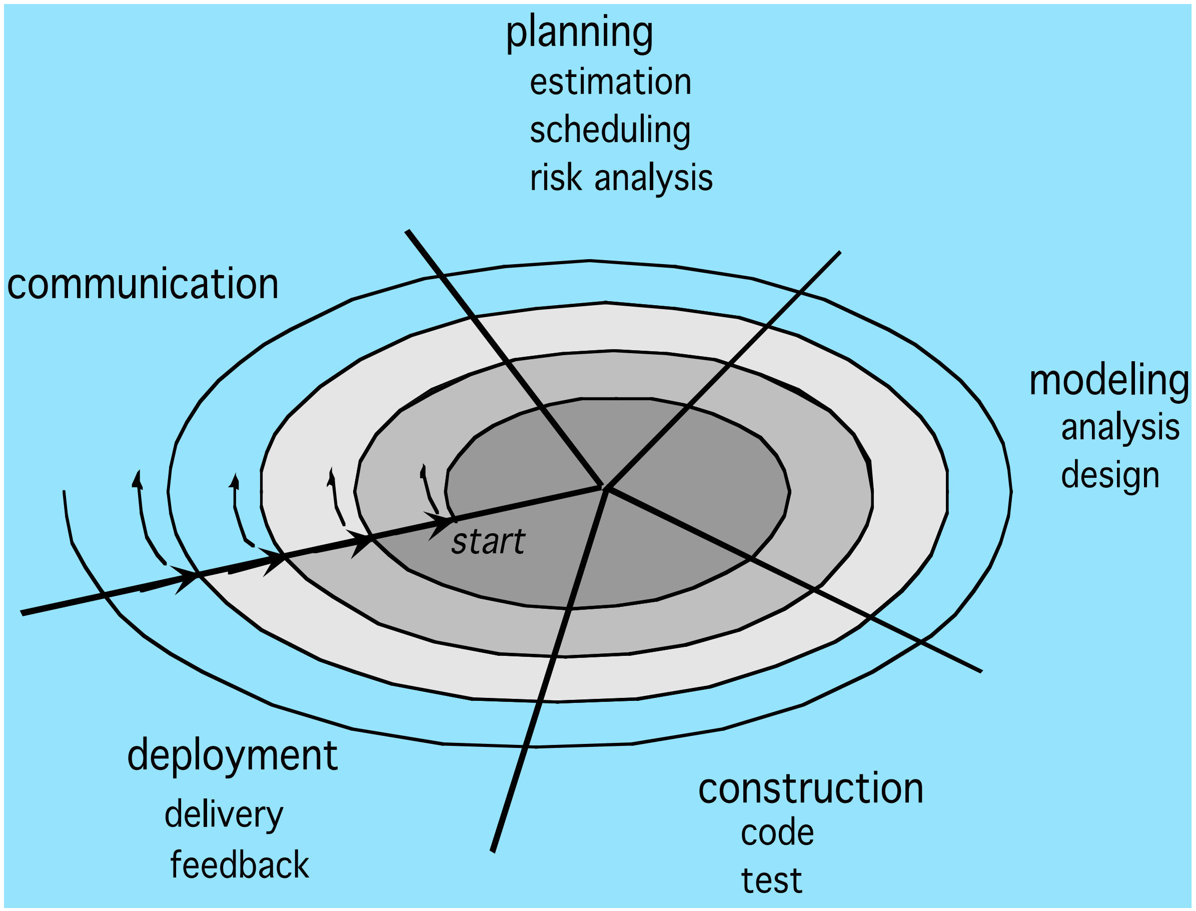
Evolutionary Models

- ✧ Iterative + Incremental
- ✧ Usually a set of core product or system requirements is well understood, but the details and extension have yet to be defined.
- ✧ It is **iterative** that enables you to develop **increasingly** more complete version of the software i.e. evolution of software.
- ✧ Two Types: **Prototyping** and **Spiral**.

The Spiral Model

- ✧ Iterative (multiple passes over the circuits) + Incremental (Each circuit adds next level to development) + Risk Driven
- ✧ Based on Water fall model (Same steps in sequence)
- ✧ The first circuit in the clockwise direction might result in the product specification; subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software.
- ✧ Each pass results in adjustments to the project plan. Cost and schedule are adjusted based on feedback. Also, the number of iterations will be adjusted by project manager.

The Spiral Model



Phases –Spiral Model

- **Task set:** In this model each of the regions with the set of work tasks is called as task set. The size of the task set will vary according to the project.
- **Task region:** Task region is a region where set of task is achieved.
- **Principle of spiral model:**
  - Spiral model terminate a project, if it is too risky.
- **Customer communication:**
  - Task requires establishing effective communication between developer and customer.
- **Planning**:
  - Task requires defining resources, time, & other related information.

Phases –Spiral Model

- **Risk information:**
  - Task requires to access both technical and management risk.
- **Engineering:**
  - Task required building one or more representation of the application.
- **Construction &release:**
  - The task requires constructing the project & releasing the project to the customer.
- **Customer evaluation:**
  - Task required obtaining customer feedback based on evolution of the software representation created during the installation stage.

Advantages and Disadvantages

✧ Advantages

- User will be able to see the project development cycle.
- Risk analysis which resolves higher priority error.
- Project is very much refined.
- Reusability of the software.

✧ Disadvantages

- It is only suitable for large size project. Model is more complex to use.
- Management skill is necessary so as to analyze the risk factor.

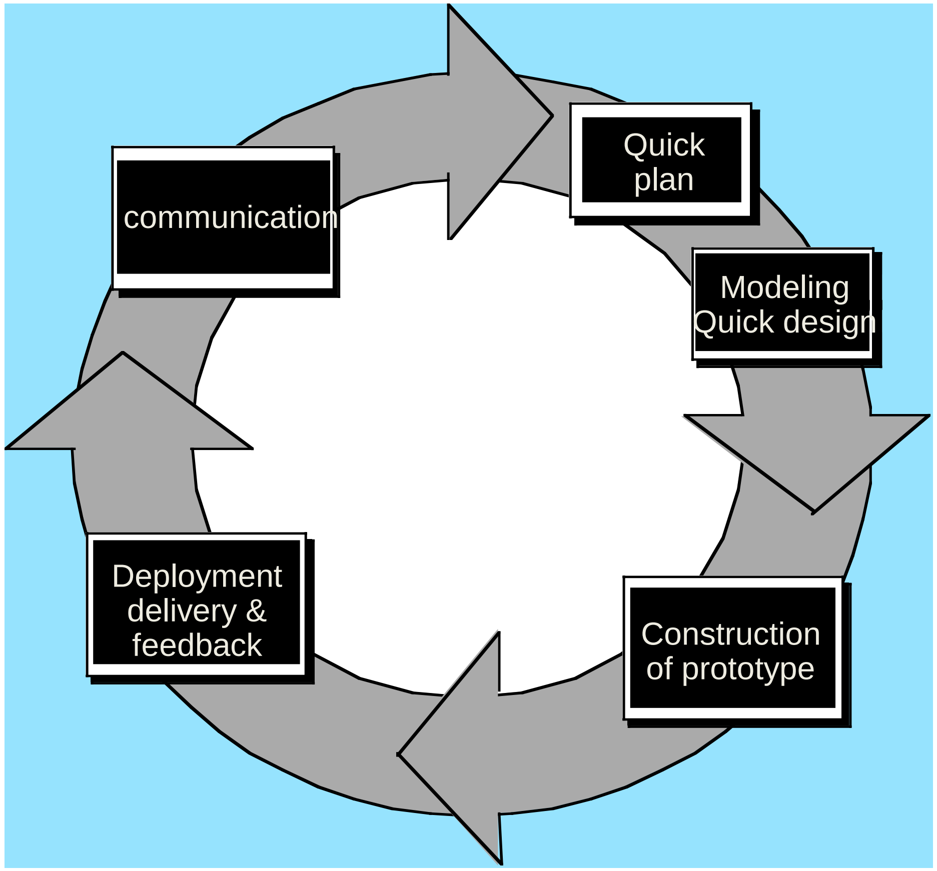
The Prototyping Model

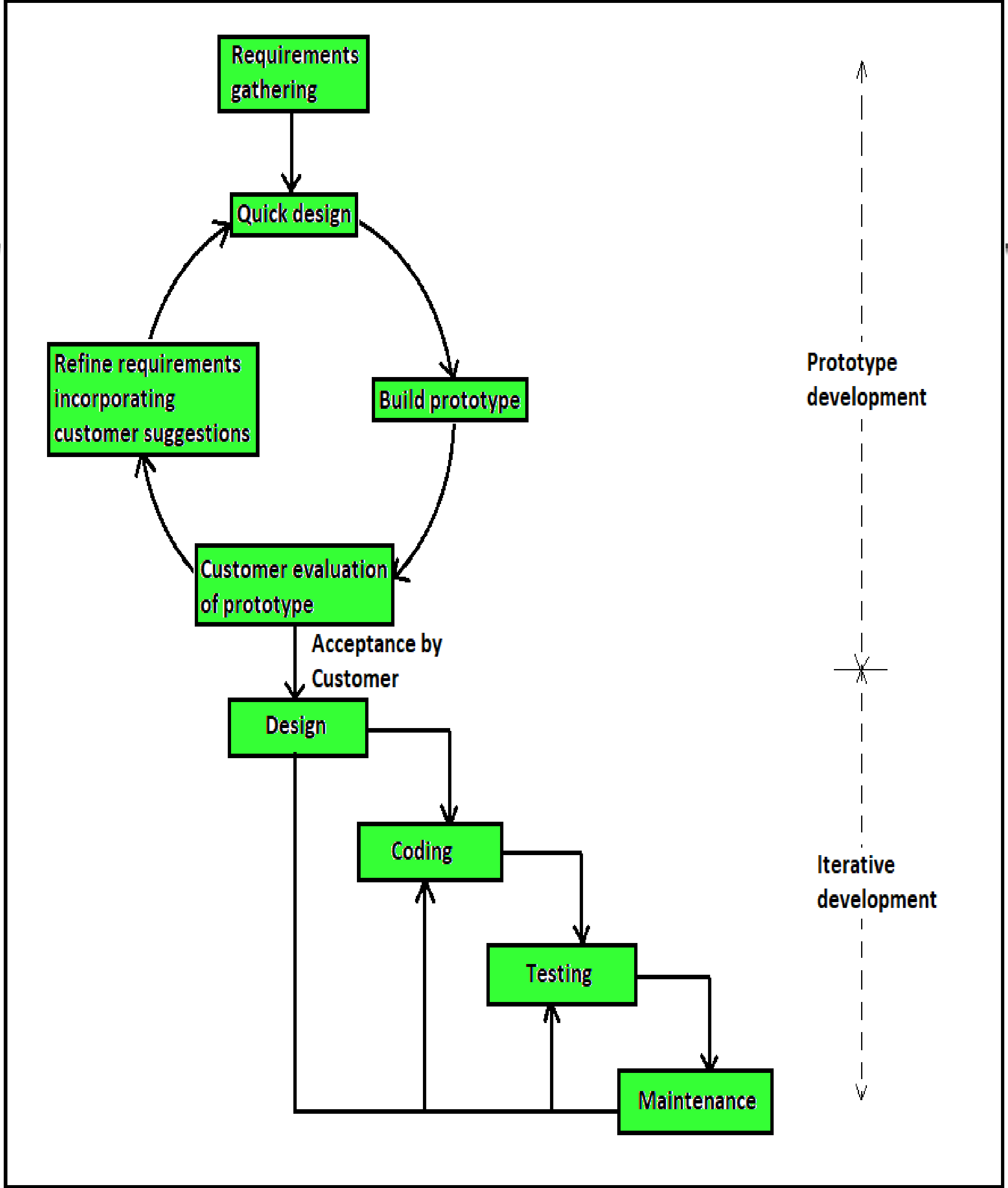
- Prototype model focuses on producing software product quickly.
- A prototype paradigm may be the best approach in many situations.
- The developer may be ensured with the efficiency of the algorithm.
- The prototyping model begins with communication.
- In this phase the software engineers and customer must define the overall objective for the software and identify whatever requirements are known, outline the areas where further definition is mandatory.

The Prototyping Model

- ✧ Often, a customer defines a set of general objectives for software, but does not identify detailed requirements for functions and features. In other cases, the developer may be unsure of the efficiency of an algorithm.
- ✧ A prototyping iteration is planned quickly, and modeling (in the form of a “quick design”) occurs.
- ✧ A quick design focuses on a representation of those aspects of the software that will be visible to end users (e.g., human interface layout or output display formats). The quick design leads to the construction of a prototype.
- ✧ The prototype is deployed and evaluated by stakeholders, who provide feedback that is used to further refine requirements.

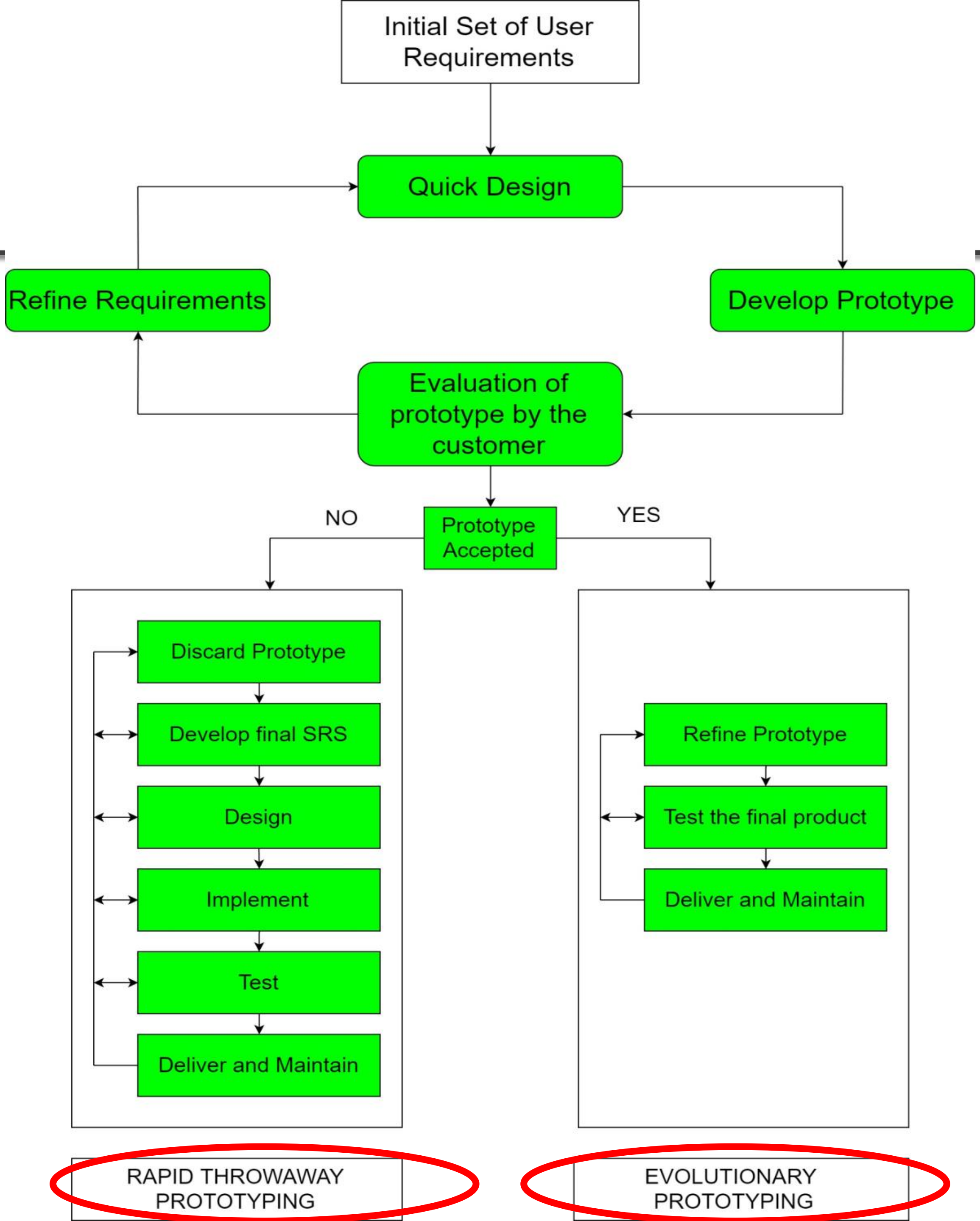
The Prototyping Model




Requirements gathering
Quick design
Build prototype
Refine requirements incorporating customer suggestions
Customer evaluation of prototype
Acceptance by Customer
Design
Coding
Testing
Maintenance
Prototype development
Iterative development

Software Engineering
Ian Sommerville

Software Engineering
Ian Sommerville


Initial Set of User Requirements
Quick Design
Refine Requirements
Develop Prototype
Evaluation of prototype by the customer
NO
Prototype Accepted
YES
Discard Prototype
Develop final SRS
Design
Implement
Test
Deliver and Maintain
Refine Prototype
Test the final product
Deliver and Maintain
RAPID THROWAWAY PROTOTYPING
EVOLUTIONARY PROTOTYPING

The Prototyping Model

- The prototype can serve as “the first system.” The one that Brooks recommends you throw away. But this may be an idealized view. Although some prototypes are built as “throwaways,” others are evolutionary in the sense that the prototype slowly evolves into the actual system.

- The key is to define the rules of the game at the beginning; that is, all stakeholders should agree that the prototype is built to serve as a mechanism for defining requirements. It is then discarded (at least in part), and the actual software is engineered with an eye toward quality.

Problem with the prototyping model

- ✧ Stakeholders see what appears to be a working version of the software, unaware that the prototype is held together haphazardly, unaware that in the rush to get it working you haven't considered overall software quality or long-term maintainability.
- ✧
- ✧ You often make implementation compromises in order to get a prototype working quickly. An inappropriate operating system or programming language may be used simply because it is available and known; an inefficient algorithm may be implemented simply to demonstrate capability. After a time, you may become comfortable with these choices and forget all the reasons why they were inappropriate.

Advantages and Disadvantages

- **Advantages**
  - Iteration process facilitating enhancements.
  - Partial product can be viewed by the customer.
  - Flexibility of the product.
  - Customer satisfaction of the product.

- **Disadvantages**
  - No optimal solution.
  - Time consuming if algorithm used is inefficient.
  - Poor documentation resulting with difficulty.

3 Concerns on Evolutionary Process

1. Planning is at Risk in prototyping model due to uncertain no: of cycles required to reach the final product.

2. Development speed is some what slow as the iterations need time to cover the improvements. Hence being too fast will bring the process to chaos and being too slow will effect the productivity.

3. High Quality VS Timely delivery tradeoff. Iterations do improve but to what extent iterations are to be added is a question.

RAD Model (Rapid Application Development)

- ✧ RAD is an incremental software process models that emphasis a short development cycle
- ✧ In RAD model the rapid development is achieved by using a component based construction approach.
- ✧ The first phase is communication phase it works to understand the business problem & information characteristics.
- ✧ Planning phase is essential because multiple software teams work in parallel on different software function.

RAD

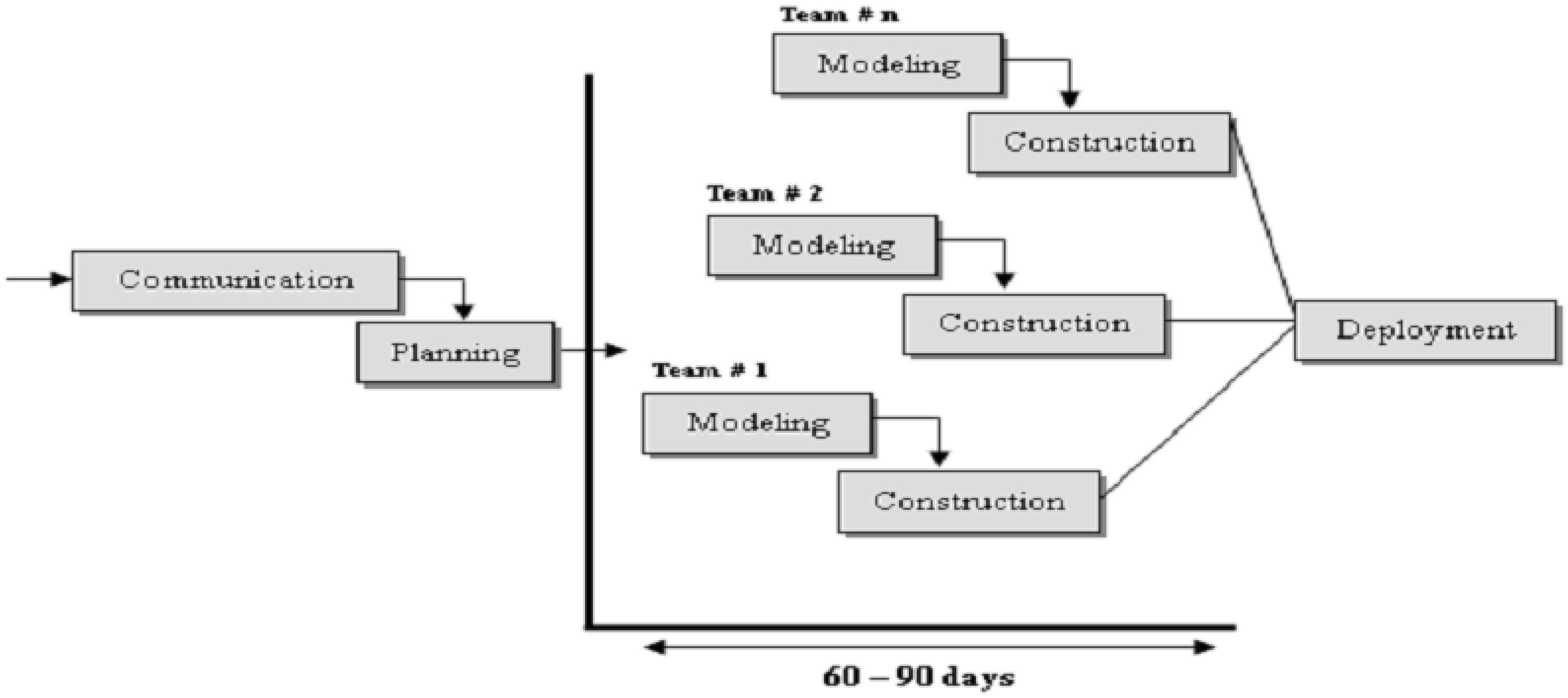


Fig: 1.8 RAD model

Drawbacks of RAD

- For large but scalable projects RAD requires sufficient human resources to create the right number of RAD team.
- If developer and customer are not committed to the rapid-fire activities necessary to complete the software in a much abbreviated time frame, RAD project will fail.
- If a system cannot be properly modularized, building the component necessary for RAD will be problematic.
- If high performance is an issue & performance is to be achieved through tuning the interface to system component, the RAD approach may not work.
- RAD may not be appropriate when technical risks are high. Eg: when a new application makes necessary use of new technology.

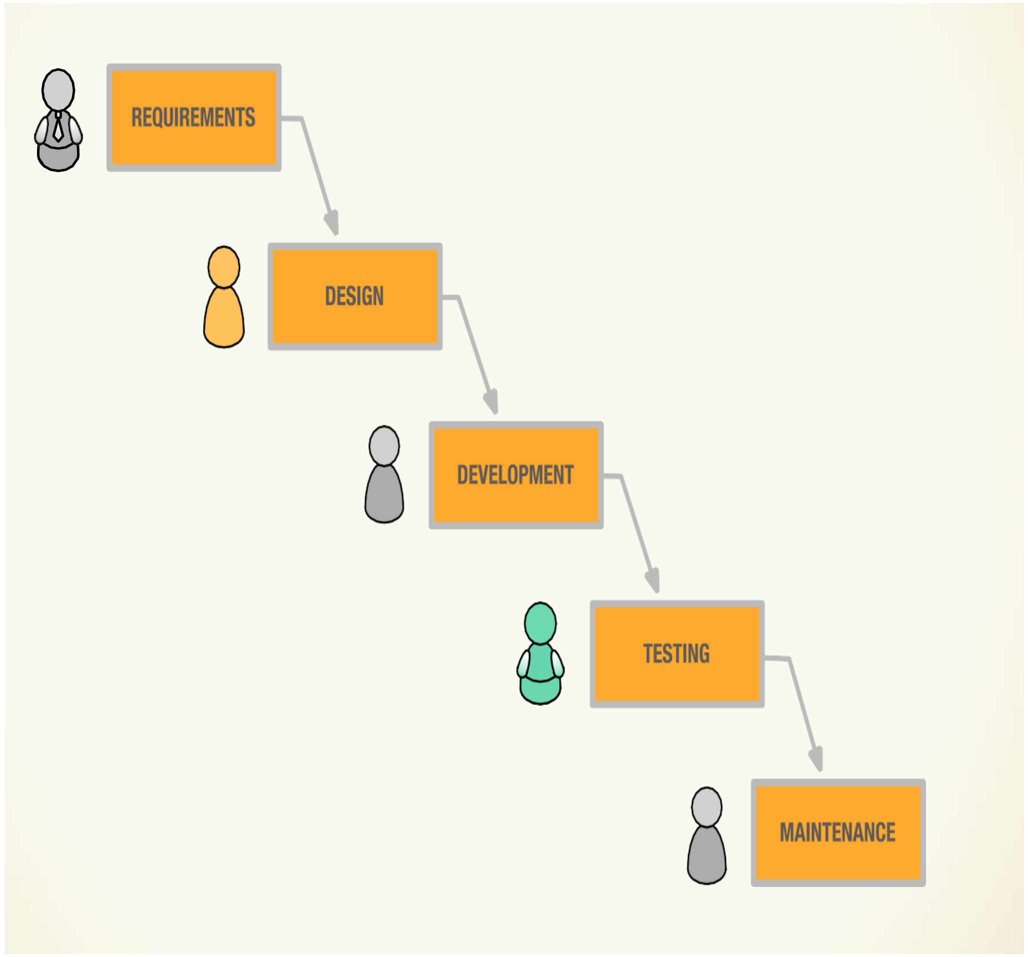
Advantages and Disadvantages

- **Disadvantage**
  - Difficult to divide problem into several unit.
  - Used only for very large projects.

- **Advantage**
  - Reduce errors

Agile Development

Remember waterfall?



Agile Manifesto

- **"We are uncovering better ways of developing software by doing it and helping others do it.**
- Through this work we have come to value:
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- **That is, while there is value in the items on the right, we value the items on the left more."**

Kent Beck et al

Agile Development

- ✧ **Agile development methods** emerged in the late 1990s whose aim was to radically reduce the delivery time for working software systems:
- ✧ Program specification, design, and implementation are **interleaved**
- ✧ The system is developed as a series of frequent versions or **increments**
- ✧ **Stakeholders** involved in version specification and evaluation
- ✧ Extensive **tool support** (e.g. automated testing tools) used to support development
- ✧ **Minimal documentation** - focus on working code

The **principles** of agile methods:

- ✧ **Customer involvement**
  - Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
- ✧ **Incremental delivery**
  - The software is developed in increments with the customer specifying the requirements to be included in each increment.
- ✧ **People not process**
  - The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.

The **principles** of agile methods:

✧ **Embrace change**

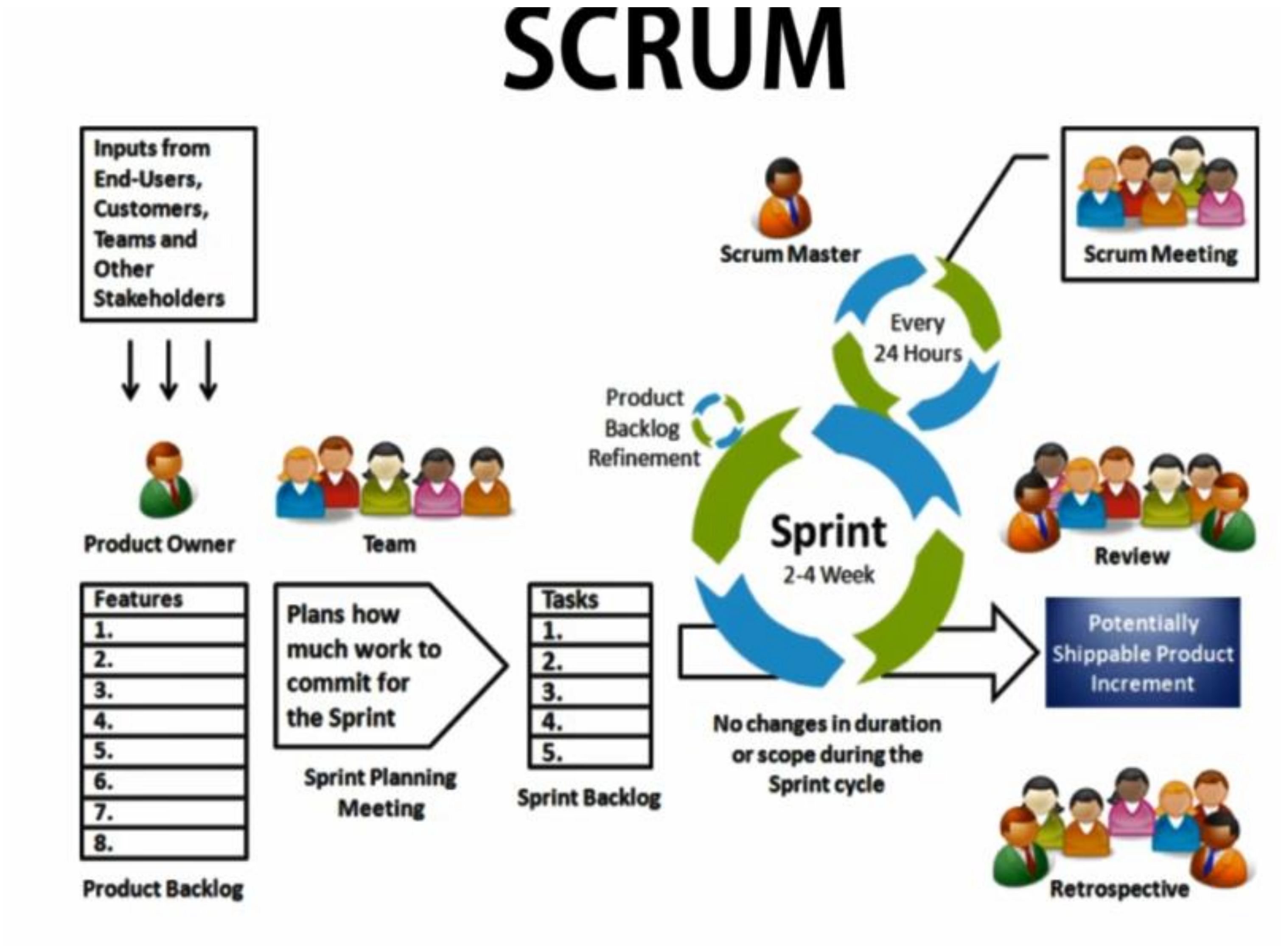
- Expect the system requirements to change and so design the system to accommodate these changes.

✧ **Maintain simplicity**

- Focus on simplicity in both the software being developed and in the development process.
- Wherever possible, actively work to eliminate complexity from the system.

Scrum Process Model

- The Scrum approach is a general agile method but its focus is on managing iterative development rather than specific agile practices. There are three phases in Scrum:
- The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
- This is followed by a series of **sprint** cycles, where each cycle develops an increment of the system.
- The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.


SCRUM
Inputs from End-Users, Customers, Teams and Other Stakeholders
Product Owner
Team
Scrum Master
Scrum Meeting
Every 24 Hours
Product Backlog Refinement
Sprint
2-4 Week
Review
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
No changes in duration or scope during the Sprint cycle
Potentially Shippable Product Increment
Retrospective

Components of Scrum

- ✧ Scrum Roles
- ✧ The Process/Events
- ✧ Scrum Artifacts

Scrum Roles

✧ The Scrum Team

✧ Scrum Master

✧ Product Owner

The Scrum Team

- ✧ Scrum teams are typically consist of 3-9 members and have no any team leader to assign tasks or decide how a problem is solved.
- ✧ The Development Team is a self-organizing, cross-functional group equipped with all kinds of the skills to deliver increments at the completion of each sprint. (such as QA, Programmers, UI designers, etc.)
- ✧ Members should be full-time.

Scrum Master

- ✧ Represents management to the project.
- ✧ No hierarchical authority over the team but rather more of a facilitator.
- ✧ Responsible for enacting scrum values and practices.
- ✧ Main job include removing impediments, facilitating meetings.

Product Owner

- The Product Owner is the one of project's key stakeholder – can be an internal or external customer, or a spokesperson for the customer. Act like one voice (in any case).

- The main responsibility of Product Owner is to convey the overall mission and vision of the product which the team is building. Product Owner have complete idea what need to be build and in what order this should be done.

- Ultimately accountable for managing the product backlog and accepting completed increments of work.

The Process/Events

- ✧ Sprint Planning Meeting
- ✧ Sprint
- ✧ Daily Scrum
- ✧ Sprint Review Meeting
- ✧ The Retrospective

Sprint

- ✧ Sprints are usually 2-4 weeks long and can be as short as one week, during which is incremented a product functionality.
- ✧ A sprint is a time-boxed period where the specific work is finished and made ready for review.
- ✧ NO outside influence can interference with the Scrum team during the Sprint.
- ✧ Each Sprint begins with the Daily Scrum Meeting.

Daily Scrum

- ✧ In a short (15 minutes long) meeting, which is held every day before the Team starts working.

- ✧ Each team member quickly and transparently discuss the progress since the last stand-up, planned work before the next meeting. Any impediments that may be blocking his or her progress is also discussed.

- ✧ Participants: Scrum Master (which is the chairman), Scrum Team.

Sprint Review Meeting

- ✧ The Sprint Review is the "show-and-tell" or demonstration event for the team. Business functionality which was created during the sprint is demonstrated to the Product Owner. It held at the end of each Sprint.
- ✧ The Product Owner analyzes the work against pre-defined acceptance criteria and either accepts or rejects the work.
- ✧ The client provides feedback to ensure that the delivered increment met the business need.

The Retrospective

- ✧ The Retrospective is the final team meeting in the Sprint to determine what went well, what didn't go well, and how the team can improve in the next Sprint.

- ✧ Attended by the team and the ScrumMaster, the Retrospective is an important opportunity for the team to focus on its overall performance and identify strategies for continuous improvement on its processes.

Scrum Artifacts

- ✧ Product Backlog
- ✧ Sprint Backlog

Product Backlog

- The product backlog is an important document that describe every requirement for a system, project or product expressed as a prioritized list of Backlog items.
- The product backlog can be considered as a to-do list consisting of work items, each of which produces a deliverable with business value.
- Backlog items are ordered in terms of business value by the Product Owner. It is managed and owned by a product Owner.
- Spreadsheet (typically).
- Usually is created during the Sprint Planning Meeting.

Sprint Backlog

- ✧ A sprint backlog is the specific list of items taken from the product backlog which are to be completed in a sprint.
- ✧ A subset of Product Backlog Items, which define the work for a Sprint.
- ✧ Is created ONLY by Team members.
- ✧ Each Item has its own status.
- ✧ Should be updated every day.

Sprint Backlog

- ✧ No more than 300 tasks in the list.
- ✧ If a task requires more than 16 hours, it should be broken down.
- ✧ Team can add or subtract items from the list. Product Owner is not allowed to do it.