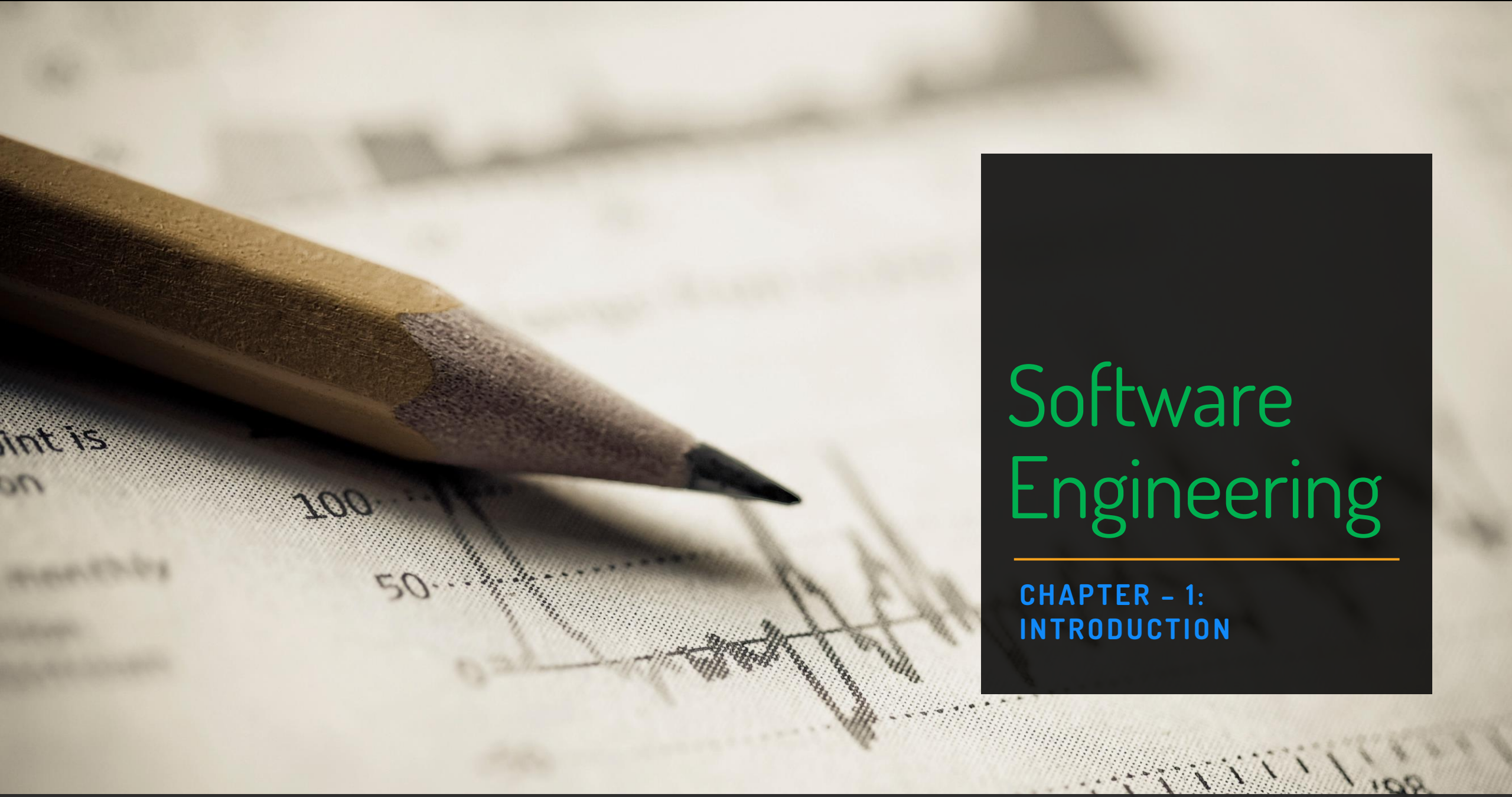


Software Engineering

CHAPTER – 1: INTRODUCTION

Outline

- Introduction
- Two Orthogonal view of software
- The Software engineering?
  - Shifting the Paradigm
- Software Engineering Activities
- Software development process models
- Object oriented system development methodology
  - Why an object oriented
  - Overview of the unified approach.
  - An object-oriented philosophy
  - Basic concepts of an object
  - Attributes of an object, its state and properties

Introduction

Software can be defined as a **set of programs** and **associated documentations** such as the following:

- **A number of separate programs**
- **Configuration files**
- **System documentation**
- **User documentation**

Introduction

- ***Software development*** refers to all **activities** that go into producing an information systems solution. The activities include:
  - **System analysis**
  - **Modeling**
  - **Design**
  - **Implementation**
  - **Testing and**
  - **Maintenance**

Introduction

- Developing software can be a complicated process, having the following *activities:*
  - Problem definition
  - Requirements development
  - Construction planning
  - Software architecture, or high-level design
  - Detailed design
  - Coding and debugging
  - Unit testing
  - Integration testing
  - Integration
  - System testing
  - Corrective maintenance

Two Orthogonal View of Software Development

Traditional Technique –*focuses on* **functions, what it is doing**

- **Software as a collection of programs(functions) and isolated data**
- **Program = Algorithms + Data Structures**
- **A software system is a set of mechanisms for preforming certain action on certain data**

- ***Object Oriented methodologies*** –*focuses on* **objects** *that* **combines data and functionality**

Two Orthogonal View of Software Development

TRADITIONAL APPROACH	OBJECT ORIENTED SYSTEM DEVELOPMENT
Collection of procedures(functions)	Combination of data and functionality
Focuses on function and procedures, different styles and methodologies for each step of process	Focuses on object, classes, modules that can be easily replaced, modified and reused.
Moving from one phase to another phase is complex.	Moving from one phase to another phase is easier.
Increases duration of project	decreases duration of project
Increases complexity	Reduces complexity and redundancy

Software Engineering?

- The term software engineering was coined in 1968 as a response to the **desolate state** of the art of developing **quality software on time** *and* **within budget.**
- Software developers were **NOT ABLE TO SET concrete objectives, predict the resources necessary to attain those objectives, and manage the customers' expectations**
- ***Software engineering*** *as a concept was created at the end of the 1960s. The term was first used by* **Margaret Hamilton --** *director of the Software Engineering Division of the MIT Instrumentation Lab. Margaret was leading the effort to develop* the **flight-control software for the Apollo space program.**

Software Engineering?

- *During the same period, the **NATO convened a conference** in Garmisch Partenkirchen, Germany, to try to define the term.* This was the first software engineering conference.
- *A conclusion of the conferees was that **software engineering should use the philosophies and paradigms of established engineering disciplines** to solve what they termed the **software crisis** , namely, that the **quality of software generally was unacceptably low** and that **deadlines** and **budgets** were not being met.*
- *The earliest computers had been programmed by* **flipping switches, or even hard-coded as part of their design.** *This was **slow and inflexible**, and the idea of the* **"stored program"** *was born.* **This is the idea that, for the first time, made a clear distinction between software and hardware.**

Software engineering? *Shifting the Paradigm*

- **The idea of paradigm shift was created by physicist Thomas Kuhn.**
- *Most learning is a kind of* **accretion**. ***We build up layers of understanding, with each layer foundationally under-pinned by the previous one.***
- *However, not all learning is like that.* **Sometimes we fundamentally change our perspective on something, and that allows us to learn new things**, ***but that also means we must discard what went before.***
- ***In the 18th century, reputable biologists believed that some animals*** **spontaneously generated themselves.** **Darwin** ***came along in the middle of the 19th century and described the*** **process of natural selection**, ***and this overturned the idea of spontaneous generation completely.***

Software engineering? *Shifting the Paradigm*

- *This* **change in thinking** *ultimately led to our* **modern understanding of genetics and our ability to understand life at a more fundamental level**, *create technologies that allow us to manipulate these genes, and create COVID-19 vaccines and genetic therapies.*

- *Similarly,* **Kepler, Copernicus, and Galileo** *challenged the then conventional wisdom that* ***Earth was at the center of the universe****. They instead proposed a* ***heliocentric model for the solar system.*** *This ultimately led to* **Newton** *creating* ***laws of gravitation*** *and* **Einstein** *creating* ***general relativity****, and* ***it allowed us to travel in space and create technologies like GPS.***

Software engineering? *Shifting the Paradigm*

- *The idea of paradigm shift implicitly includes the idea that* **when we make such a shift, we will, as part of that process, discard some other ideas that we now know are no longer correct.**
- *The implications of treating* ***software development as a genuine engineering discipline****, rooted in the philosophy of the scientific method and scientific rationalism, are* **profound**.
- *It is* **profound** *not only in its,* ***impact and effectiveness****, but also in the essential need to discard the ideas that this approach supersedes. This gives us an approach to* ***learning more effectively and discarding bad ideas more efficiently***.
- **The paradigm shift** ***(engineering thinking to software development)*** *in thinking differently about* ***what it is that we do, and how we do it****, when we create software should* ***help us make SW development simpler, more reliable, and more efficient.***
- *This is* **not about more bureaucracy***; it is about* **enhancing our ability to create high-quality software more sustainably and more reliably.**

Engineering?

- The emphasis in software engineering is on both words, **software** and **engineering.**
- **Engineering—The Practical Application of Science**
- **Engineering is the application of an empirical, scientific approach to finding efficient, economic solutions to practical problems.**
- **Engineering solutions are not abstract ivory-tower things**; they are **practical and applicable** to the problem and the context.
- They are efficient, and they are created with an understanding of, and constrained by, **the economics of the situation.**
- **Engineering is about applying scientific rationalism to solving problems.**. It is the **processes, tools, *and* techniques. It is the ideas, philosophy, and approach that together make up an engineering discipline.**
- An engineer is able to build a high-quality product using **off-the-shelf components** and integrating them **under time and budget constraints**.

Engineering?

- The engineer is often faced with ill-defined problems, partial solutions, and has to rely on **empirical methods** to evaluate solutions.
- Engineers working on application domains such as passenger aircraft design and bridge construction have met successfully similar challenges.
- **BUT,**
- **Software engineers have not been as successful.**
- For the "**failure of software engineering**," the **programmers are to be blamed and not their managers**, because, **it is a failure in the whole approach to producing software**. **The bad planning,, the bad culture, the bad code (lots of bugs apparently).**

Software Engineering?
Problems

- The problem of **building** and **delivering** complex software systems **on time** has been actively investigated and researched.
- Everything has been blamed, from the customer (“***What do you mean I can’t get the moon for $50?***”) to the “soft” in software (“***If I could add that one last feature …***”).

  What is the problem?
- **Complexity and change:** Useful software systems are complex. To remain useful they need to evolve with the end users’ need and the target environment.

Software Engineering?
Problems

- Software systems are complex creations: They perform **many functions**; they are built to achieve **many different**, and **often conflicting, objectives**;
  - they comprise **many complex and custom made components**;
  - **many participants**, from different disciplines, take part in the development of these components;
  - the development process and **the software life cycle often spans many years**;
  - finally, complex systems are **difficult to understand completely by any single person**.
  - Many systems are so hard to understand, even during their development phase, that they are never finished: These are called **vaporware**.

Software Engineering?
Problems: *Constant Change*

- Software development projects are subject to **constant change**.
- Because **requirements are complex**, they need to be **updated when errors are discovered** and **when the developers have a better understanding of the application**.
- If the project lasts many years, the staff turn-out is high, requiring constant training. **The time between technological changes is often shorter than the duration of the project.**
- The widespread assumption of a software project manager ***that all changes have been dealt with and that the requirements can be frozen*** **lead to an irrelevant system being deployed.**

Software Engineering Vs Other Engineering

"In other disciplines,* engineering *simply means the* "stuff that works." *It is the* process *and* practice *that you apply to increase your chances of doing a good job."

- Software development is a process of **discovery** and **exploration**; therefore, to succeed at it, software engineers need to become experts **at learning.**
- **Humanity's best approach to learning is science**, so we need to adopt the **techniques and strategies** of science and apply them to our problems. ***This is often misunderstood to mean that we need to become physicists measuring things to unreasonable, in the context of software, levels of precision***.
- ***Engineering is more pragmatic:*** dealing with things ***sensibly*** and ***realistically*** in a way that is based on **practical rather than theoretical considerations.**

Software Engineering: Definitions

- ***Software engineering*** is about **production of fault-free software, delivered on time and within budget, that satisfies the client's needs**.
- ***Software Engineering*** is the **construction of quality software** with in the **context of constant change.**
- ***Software engineering*** is a **detailed study of engineering** to the **design, development and maintenance of software.**
- ***Software engineering is To organize and control software development process and produce a well-structured***, **accurate** and **useful** software solution.

Software Engineering: Definitions

- Software is built to meet a certain **functional goal and satisfy certain qualities.** *The fundamental assumptions of software engineering are:* ***Good processes lead to good software*** **and** ***Good processes reduce risk***
- ***Software engineers*** *need to become experts at* ***managing complexity.*** ***Whatever the nature of the problems that software engineer solve or the technologies that he/she use to solve them,*** **addressing the complexity of the problems that face him/her and the solutions that we apply to them is a central differentiator between bad systems and good.**
- ***Software engineer*** *has to acquire a broad range of* **skills**, *both* **technical** *and* **managerial**. *These skills have to be applied not just to programming but to* **every step of software production**, *from* **requirements** *to* **postdelivery maintenance.**

Software Engineering Activity: *Modeling*

- *The* **purpose of sciences** *is to* **describe** *and* **understand complex systems** *such as:* ***system of atoms, solar systems or society of human beings.*** *Traditionally, we have:*
  - **Natural science:** ***physics, biology, chemistry, …***
  - **Social science:** ***Sociology , psychology, …***
  - *We have another system:* ***Artificial system:*** **space shuttle, airline reservation systems, and stock trading systems**
- **Herbert Simon** *coined the term* ***sciences of the artificial*** *to describe the* **sciences that deal with artificial systems**
  - **Computer science,** *for example,* ***the science dealing with understanding computer systems,*** *is a child of this century.*
- *One of the* **basic methods of science** *is* **modeling**

Software Engineering Activity: *Modeling*

- *A* **Model** *is an* **abstract representation of a system** *that enables us to answer questions about the system.*
- **Models** *are useful when dealing with* **systems that are too large, too small, too complicated, or too expensive to experience firsthand**.
- **Models** *also allow to* **visualize** *and* **understand** *systems that either no longer exist or that are only claimed to exist.*

Software Engineering Activity: *Modeling*

- *Dinosaur: From the bone fragments, Biologists reconstruct a model of the animal, following rules of anatomy. The more bones they find, the clearer their idea of how the pieces fit together and the higher the confidence that their model matches the original dinosaur. If they find a sufficient number of bones, teeth, and claws,* they can almost be sure that their model reflects reality accurately, and *they can guess the missing parts. Legs, for example, usually come in pairs. If the left leg is found, but the right leg is missing, the fossil biologists have a fairly good idea what the missing leg should look like and where it fits in the model.* This is an example of a model of a system that no longer exists.

Software Engineering Activity: *Modeling*

- *Today's high-energy physicists are in a similar position to that of a fossil biologist who has found most of the bones. Physicists are building a model of matter and energy and how they fit together at the most basic subatomic level. Their tool is the high-energy particle accelerator. Many years of experiments with particle accelerators have given high-energy physicists enough confidence* that their models reflect reality and that the remaining pieces that are not yet found will fit into the so-called standard model. This is an example of a model for a system that is claimed to exist.

Software Engineering Activity: *Modeling*

- Both **system modelers**, **fossil biologists** and **high-energy physicists**, deal with two types of entities:
  - **The real-world system**, observed in terms of a set of phenomena, and
  - **The problem domain model**, represented as a set of interdependent concepts.
- **The system in the real world** is ***a dinosaur or subatomic particles.***
- **The problem domain model** is ***a description of those aspects of the real-world system that are relevant to the problem under consideration.***

Software Engineering Activity: *Modeling*

- **Software engineers** face similar challenges as fossil biologists and high-energy physicists.
- **First**, software engineers **need to understand the environment in which the system is to operate**.
  - For a **train traffic control system**, software engineers **need to know train signaling procedures.**
  - For a **stock trading system**, software engineers **need to know trading rules.**
  - **Software engineers do not need to become a fully certified train dispatcher or a stock broker**; **they only need to learn the problem domain concepts that are *relevant* to the system**. *In other terms*, **they need to build a model of the problem domain.**
- **Second,** software engineers **need to understand the systems they could build, to evaluate different solutions and trade-offs.**
  - **Most systems are too complex to be understood by any one person and are expensive to build.**
  - To address these challenges, software engineers describe important aspects of the alternative systems they investigate. *In other terms,* **they need to build a model of the solution domain.**

Software Engineering Activity: *Problem Solving*

- **Engineering** *is a* **problem-solving activity**. *Engineers* **search for an appropriate solution**, ***often by trial and error***, *evaluating alternatives empirically, with limited resources and incomplete knowledge. In its simplest form, the engineering method includes five steps:*

1. **Formulate the problem**
2. **Analyze the problem**
3. **Search for solutions**
4. **Decide on the appropriate solution**
5. **Specify the solution**

Software Engineering Activity: *Problem Solving*

- Software engineering *is an engineering activity. It requires* experimentation, the reuse of pattern solutions, and the incremental evolution of the system toward a solution that is acceptable to the client.
- *Software development typically includes five development activities:* requirements elicitation, analysis, system design, object design, and implementation.
- *During* requirements elicitation and analysis, *software engineers* formulate the problem *with the client and* build the problem domain model. Requirements elicitation and analysis correspond to steps 1 and 2 *of the engineering method.*
- *During* system design, *software engineers* analyze the problem, *break it down into smaller pieces, and select general strategies for designing the system. During* object design, *they* select detail solutions *for each piece and* decide on the most appropriate solution. System design and object design result in the solution domain model. System and object design correspond to steps 3 and 4 *of the engineering method.*
- *During* implementation, *software engineers realize the system by* translating the solution domain model into an executable representation. Implementation corresponds to step 5 *of the engineering method.*
- What makes software engineering different from problem solving in other sciences is that change occurs while the problem is being solved.

Software Engineering Activity: *Problem Solving*

- **Software development** *also includes activities whose purpose is to* **evaluate the appropriateness of the respective models.**
- **During the analysis review**, *the* **application domain model is compared with the client's reality**, *which in turn might change as a result of modeling.*
- **During the design review**, *the* **solution domain model is evaluated against project goals**.
- **During testing,** *the* **system is validated against the solution domain model**, *which might change due to the introduction of new technologies.*
- **During project management**, *managers* **compare their model of the development process** ***(i.e., the project schedule and budget)*** **against reality** ***(i.e., the delivered work products and expended resources)***.

Software Engineering Activity: *Knowledge Acquisition*

- **Linear** *acquisition model for knowledge or also called* **"the bucket theory of the mind" is a problem**
- **Knowledge acquisition** *is a* **nonlinear process.** *The* **addition of a new piece of information may invalidate all the knowledge we have acquired for the understanding of a system**.
- Even if we had already documented this understanding in documents and code (*"**The system is 90% coded, we will be done next week**"*), we must be mentally prepared to start from scratch. This had important implications on the set of activities and their interactions we define to develop the software system.
- The equivalent of the **bucket theory of the mind** is the linear **waterfall model for software development**, in which all steps of the engineering method are accomplished sequentially.

Software Engineering Activity: *Knowledge Acquisition*

- *There are several software processes that deal with this problem by* **avoiding the linear dependencies inherent in the waterfall model.**
- **Risk-based development** *attempts to anticipate surprises late in a project by identifying the high-risk components.*
- **Issue-based development** *attempts to remove the linearity altogether. Any development activity, be it analysis, system design, object design, implementation, testing, or delivery, can influence any other activity.*
- *In issue-based development,* **all these activities are executed in parallel. The difficulty with nonlinear development models**, *however, is that they are* ***hard to manage.***

Software Engineering Activity: *Rationale Management*

- When describing the acquisition or evolution of knowledge, we are even less well equipped than when describing the knowledge of an existing system.
- *In the field of mathematics and many other once deduction have been stated, for change to happen is very rare. "Once the axioms and the rules of deduction have been stated, the proof is timeless"*
- For software engineers, the situation is very different. Assumptions that developers make about a system change constantly.
  - Even though the application domain models eventually stabilize once developers acquire an adequate understanding of the problem, the solution domain models are in constant flux.
  - Design and implementation faults discovered during testing and usability problems discovered during user evaluation *trigger changes to the* solution models.
  - Changes can also be caused by new technology. *The availability of a long-life battery and of high-bandwidth wireless communication, for example, can trigger revisions to the concept of a portable data terminal.*

Software Engineering Activity: *Rationale Management*

- Change introduced by new technology often allows the formulation of new functional or nonfunctional requirements.
- A typical task to be done by software engineers is to change a currently operational software system to incorporate this new enabling technology.
- To change the system, it is not enough to understand its current components and behavior. It is also necessary to *capture and understand the context in which each design decision was made.* This additional knowledge is called the rationale of the system.

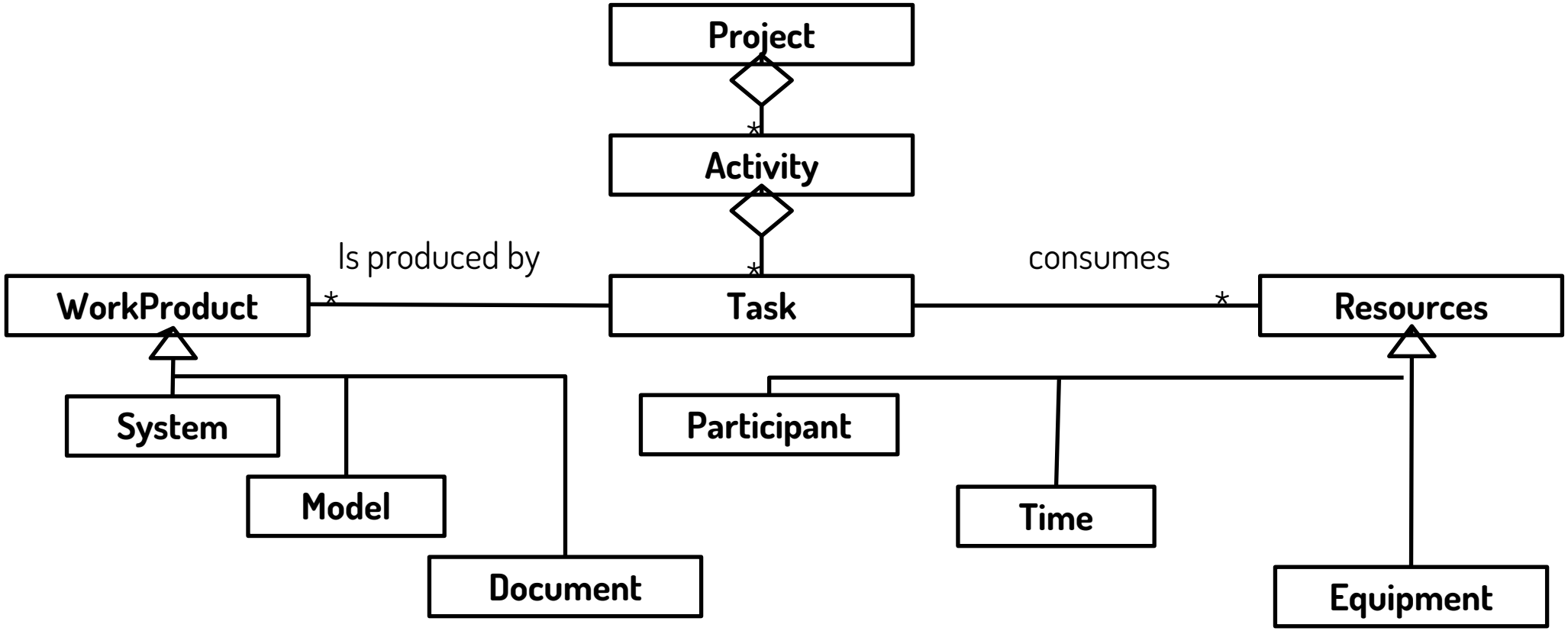
Software Engineering Activity: *Rationale Management*

- Capturing and accessing the rationale of a system is nontrivial.
- First, for every decision made, several alternatives may have been considered, evaluated, and argued. Consequently, *rationale represents a much larger amount of information than the solution models do*.
- Second, rationale information is often not explicit. *Developers make many decisions based on their experience and their intuition, without explicitly evaluating different alternatives. When asked to explain a decision, developers may have to spend a substantial amount of time recovering its rationale*.
- In order to deal with changing systems, however, software engineers must address the challenges of capturing and accessing rationale.

Software Engineering: *Concepts*

- ***Project***–set of **activities** to develop a software system
- ***Activity*** –a phase in which related **tasks** are carried out (***Requirements Elicitation, Analysis, Object Design, System Design, Implementation and Validation/testing***)
- ***Task*** –effort that uses **resources** AND **produces Work Product**
- ***Resources***–**time, equipment, people** (participants)
- ***Work Product***–a **model, system, or artifact**

Software Engineering: *Concepts*



Software Engineering:
Concepts: System and Roles

- **System**
  - A collection of interconnected parts
- **Roles**
  - **Client** – high level requirements, scope, deadline, budget, quality criteria
  - **User** – domain knowledge
  - **Manager** – progress evaluation, resource managing
  - **Developers** – construction (analyzing, designing, implementing, and testing)
  - **Technical writer** - documentation

Software Process Model

- The earlier approach (build and Fix approach)
  - Product is constructed without specification or any attempt at design.
  - Developers simply build a product that is reworked as many times as necessary to satisfy the client.
  - This model may work for small projects but is totally unsatisfactory for products of any reasonable size.
  - Maintenance is high.
  - Source of difficulties and deficiencies
    - impossible to predict
    - impossible to manage

Software Process Model: Why Models are needed?

- Symptoms of inadequacy: the software crisis
  - scheduled time and cost exceeded
  - user expectations not met
  - poor quality

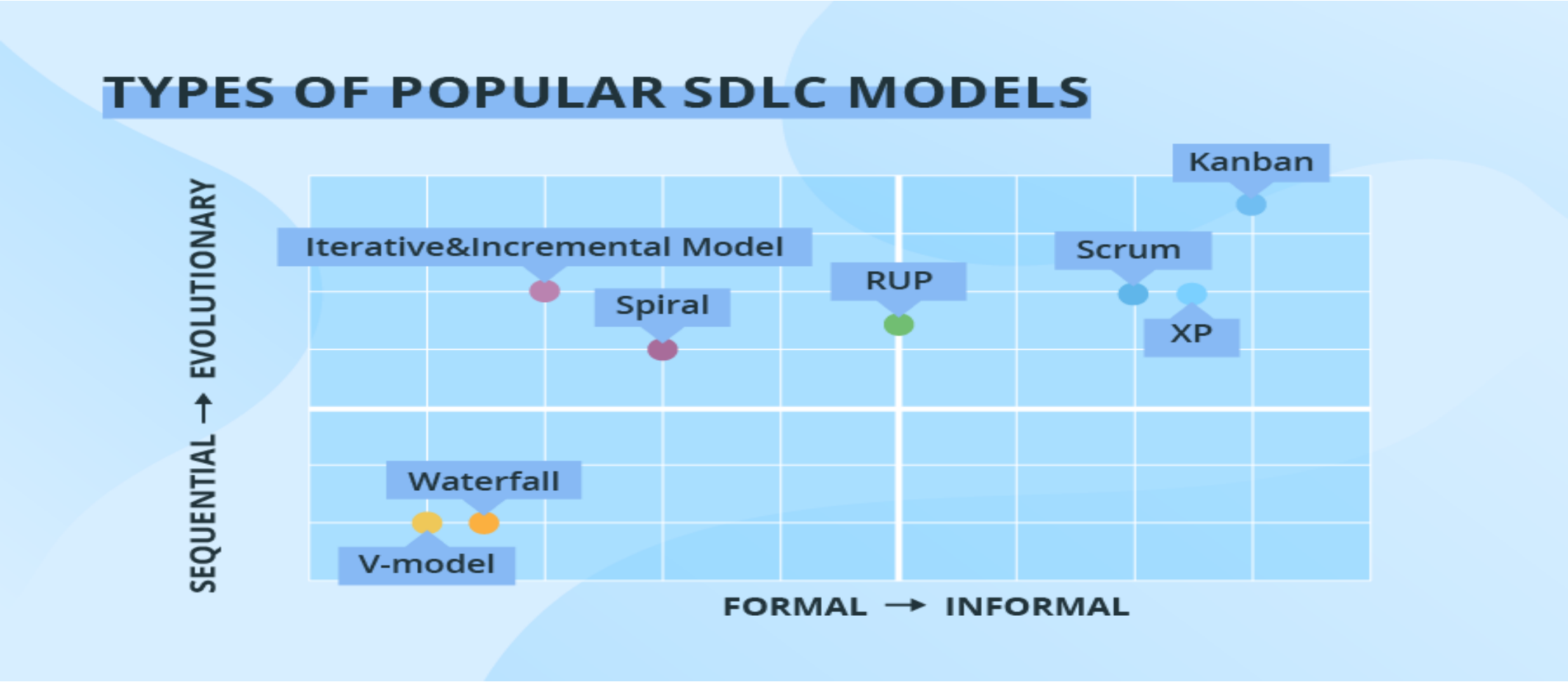
Software Process Model

- Software development life cycle (SDLC) or Software process model *show the ways to navigate through the* **complex** and **demanding process** *of software building.*
- **Process models** prescribe a ***distinct*** and ***structured set of activities, actions, tasks, milestones***, and ***work products*** required to **design and build** high quality software.
- Software models provide **stability, control,** and **organization** to a process that ***if not managed can easily get out of control***
- A **project's quality, timeframes, budget,** and **ability to meet the stakeholders' expectations** largely **depend on the chosen model.**

Software Process Model

- **All models** follow a **Series of steps** unique to its type to ensure success in the process of software development
- Today, **there are more than 50 recognized SDLC models in use**. **None of them are perfect**, and each of them **advantages** and **disadvantages**
- The **models** can be **structured** into several groups **depending on how they approach workflow organization** – **linearly or iteratively** – and what **kind of relationships** are established between the **development team** and the **customer**.

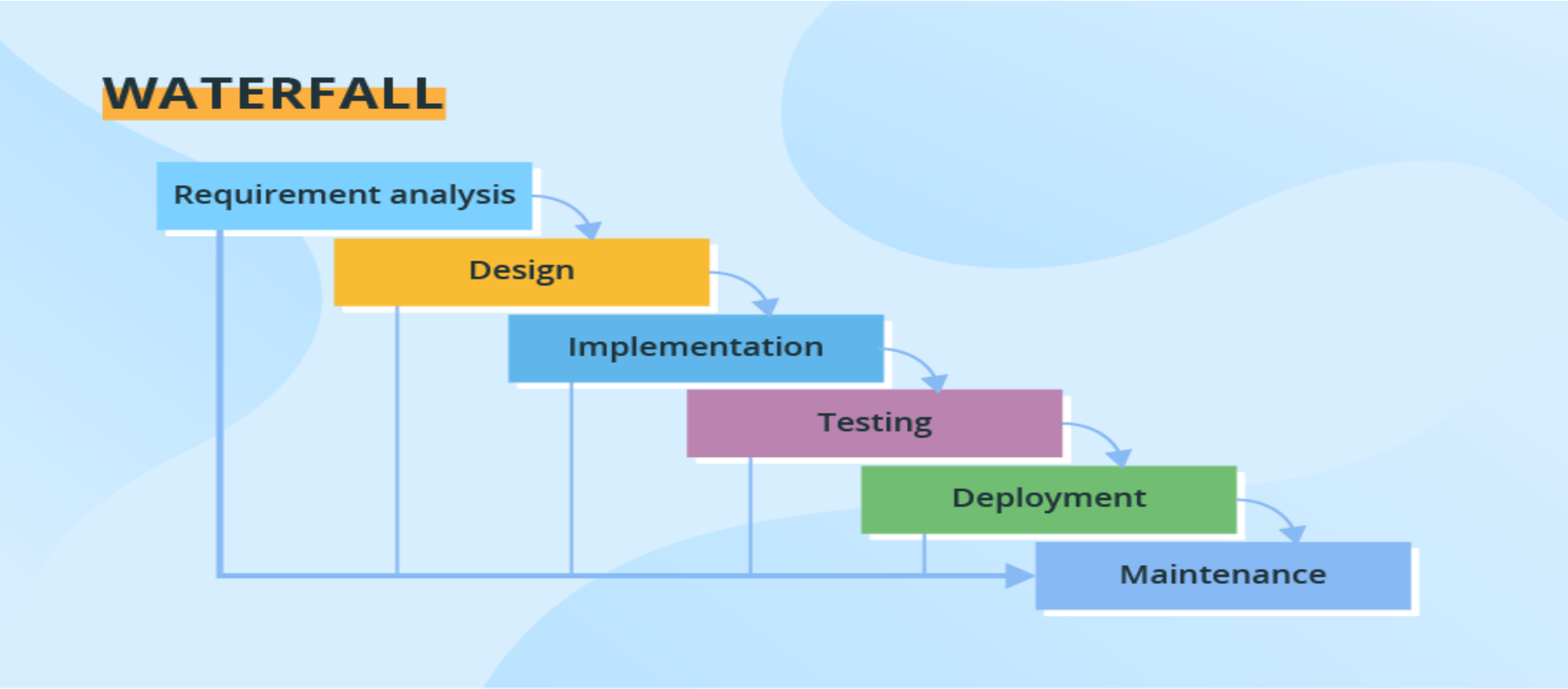
Software Process Model



Software Process Model

- The types in the **lower quadrants** of the chart take the **sequential flow.** They are **easy to implement**, **use** and **manage**.
- As you **move higher**, the **process becomes less rigid** and offers **more flexibility** when it comes to **changes in the requirements for future software.**
- The models on the **left side** of the chart imply **low customer involvement**; as you **move toward the right side**, the models grow more '**cooperative**' and **include customers** into different stages of software development life cycle more intensively.

Software Process Model:
Waterfall Model



Software Process Model: Waterfall Model

- Through all **development stages** (**analysis, design, coding, testing, deployment**), the process moves in a **cascade mode.**
- **Each stage** has concrete **deliverables** and is **strictly documented.**
- **OUTPUT of one phase becomes the INPUT of the next phase.**
- The **next stage cannot start before the previous one is fully completed (no going back to previously completed phases, no skip or on overlap of phases)**.
- **No ability to see and try software until the last development stage is finished (*all or nothing delivery*)**, which results in **high project risks** and **unpredictable project results.**

Software Process Model:

Waterfall Model: Applicability

- *Stable Requirements:* for projects which are small or mid-sized with clearly defined and unchanging requirements (small company website development).
- Projects with the need for stricter control, predictable budget and timelines (e.g., governmental projects).
- *Compliance:* It ensured strict adherence to regulations/rules and standards, especially in safety-critical and financial systems or healthcare projects.
- *Thorough Documentation:* Waterfall emphasized detailed documentation, which was crucial for industries like defense, healthcare, and banking.
- Projects where a well-known technology stack and tools are used

Software Process Model:

Waterfall Model

- Common **outputs of a waterfall**: **SRS, project plan, design docs, test plan** and **reports, final code, supporting docs**
- **Advantages:**
  - **Natural** approach for problem solving
  - Conceptually **simple** and **understandable**, cleanly divides the problem into distinct **independent phases**
  - **Easy to administer** in a contractual setup – **each phase is a milestone**
- **Problems:**

1. **Real projects are rarely to follow the *sequential model.***
2. **Difficult for the customer to *state all the requirement explicitly.***
3. **Assumes patience from customer – *working version of program will not be available until programs not getting change fully.***

Software Process Model:

Waterfall Model: Examples

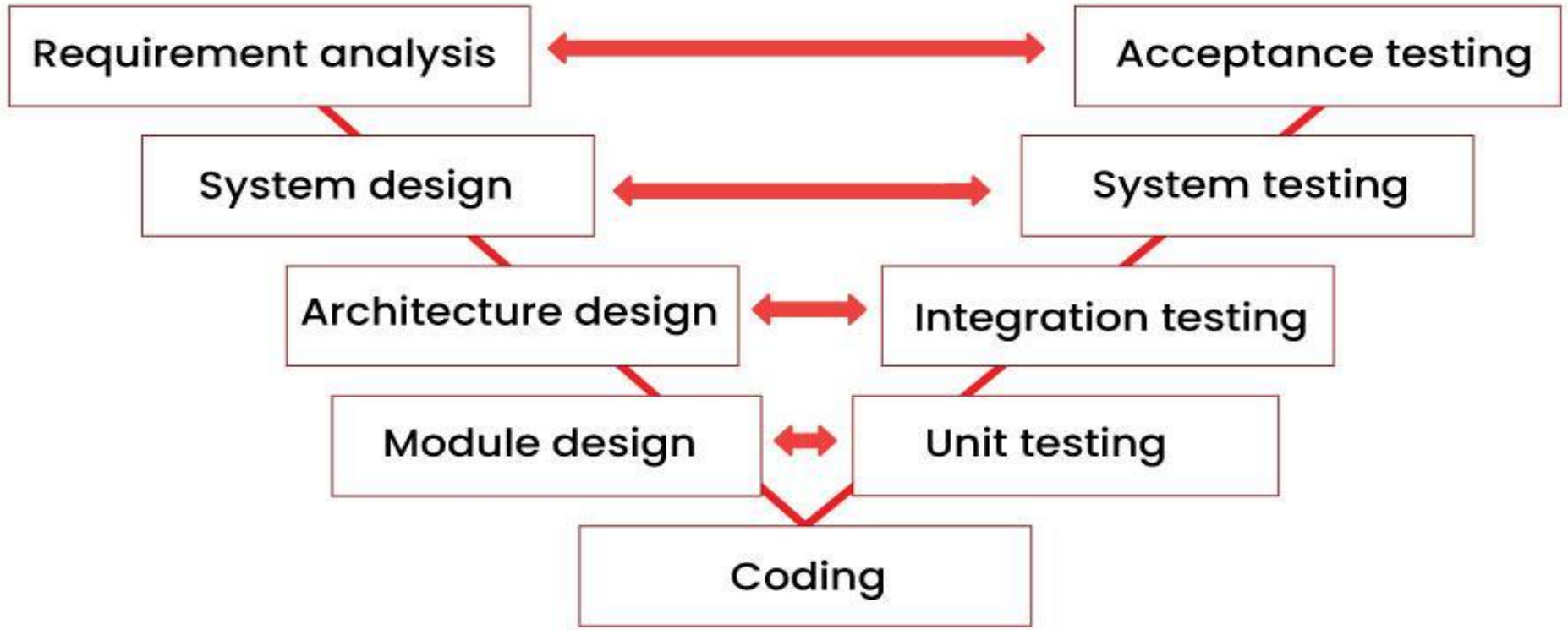
1. **Embedded Systems:**
   1. **NASA Projects:** Early space programs, such as the ***software for Apollo spacecraft and space probes***, were developed using the ***Waterfall Model*** because of the ***strict requirement specifications*** and ***zero tolerance for errors.***
   2. **Avionics Systems:** ***Flight control software*** for aircraft often used ***Waterfall*** to ensure ***reliability*** and ***compliance with safety standards.***
2. **Banking Software:**
   1. **ATM Software:** Many early ***banking systems and ATM software*** were developed using Waterfall due to the need for ***thorough documentation*** and ***structured phases.***
   2. **Payment Processing Systems:** Systems like ***SWIFT or VISA's*** early transaction systems followed ***Waterfall*** to ***ensure clear requirements*** and ***compliance with financial regulations.***
3. **Government and Defense Projects:**
   1. **Military Systems:** Projects like ***radar systems, missile control systems,*** and ***defense communication systems*** often used ***Waterfall*** because of ***rigid requirements*** and ***extensive documentation needs.***
   2. **Healthcare Software:** Early ***hospital management systems*** and ***medical imaging software*** were developed under ***Waterfall*** due to ***critical safety*** and ***regulatory standards***

Software Process Model:

Waterfall Model: Examples

4. **Enterprise Resource Planning (ERP):**
    1. **Early versions of ERP software, like SAP R/3,** were developed using ***Waterfall*** because it required ***detailed specifications*** and ***planning for organizational needs***.
5. **Telecommunications Systems: Software for early telephone switching systems**, like **Alcatel-Lucent's switching systems**, followed the ***Waterfall*** to ensure ***smooth operation*** and ***reliability***.
6. **Legacy Software:**
    1. **Operating Systems:** Early versions of **operating systems like UNIX** and some **versions of Windows** (e.g., **Windows XP**) used ***Waterfall*** during development due to the ***need for a clear design*** and ***implementation process***.
    2. **Database Systems:** Early versions of database systems like **Oracle** were developed with ***Waterfall*** to meet ***specific business and performance requirements.***

Software Process Model:
V- Model



V-Model

Software Process Model:

V- Model

- The **V-model (Verification and Validation model)** is ***(extension of the Waterfall Model)*** also **linear model** with **each stage having a corresponding testing activity** ***where new phase can start only when the current stage is complete.***
- In **V-Model** with each development phase, its testing phase is also associated in a **V-shape.**
- **Software development and testing activities take place at the same time.**
- The **left side** represents the **development activity**, the **right side** represents the **testing activity.**
- **In V-model Instead of moving down linearly** through the software development stages, **it moves down until the coding phase** and begins to **ascend upward to form a "V" shape.**

Software Process Model:
V- Model

- Is **applicable** in projects where **failures** and **downtimes** are **unacceptable (e.g., medical software, aviation fleet management software).**
- The **V-Model** is great for **smaller projects.** Using the V-Model **can yield a higher chance of success due to the test plans of the development stage.**
- **V-Model** implies **exceptional quality control, but at the same time**, it makes the V-model one of the **most expensive** and **time-consuming models**.
- Similar to the Waterfall Model, the **V-Model** is **very rigid in nature** so it isn't ideal for applications or systems software that may require **unforeseen changes/updates throughout the software lifecycle.**

Software Process Model:

V- Model: Phases

- **Requirements analysis:-** requirements of the product are analyzed **according to the customer's needs.** In this phase, **acceptance tests are designed for later use.**
- **System design:-** a **complete description of the hardware** and **all the technical components required to develop the product.**
- **Architectural design:- architectural specifications are designed**. It contains the specification of **how the software will link internally and externally with all the components**. **Therefore, this phase is also called high level design (HLD).**

Software Process Model:

V- Model: Phases

- **Module design:-** the **internal design of all the modules of the system is specified**. **Therefore, it is called low level design (LLD). All modules should be according to the system architecture**. **Unit tests are also designed in the module design phase.**

- **Coding phase:- coding of the design** and **specification done in the previous phases is done. This phase takes the most time.**

Software Process Model:

V- Model: Phases

- **Unit testing:-** the **unit tests created during the module design phase are executed**. **Unit testing is code level testing, it only verifies the technical design.** Therefore, **it is not able to test all the defects.**

- **Integration testing:-** the **integration tests created in the architectural design phase are executed**. **Integration testing ensures that all modules are working well together.**

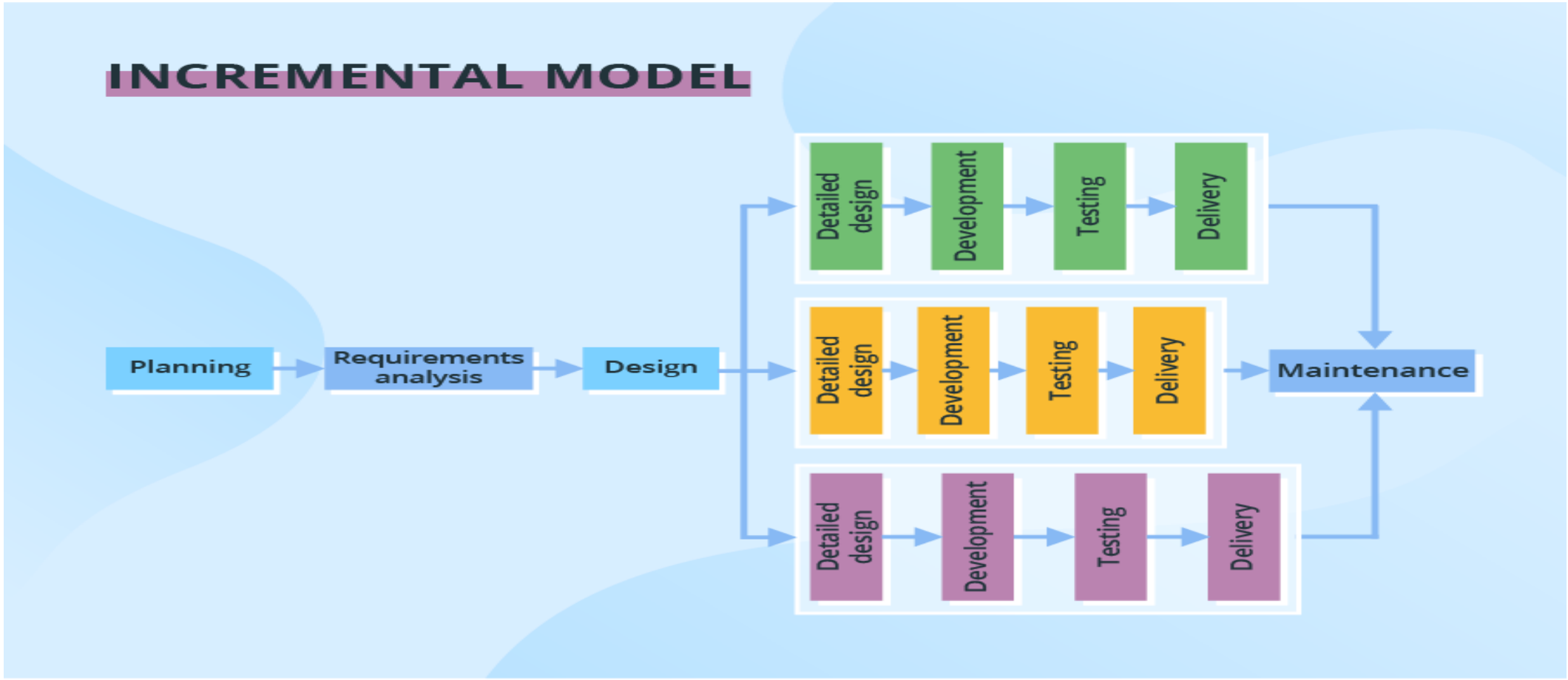
Software Process Model:

V- Model: Phases

- **System testing:–** the **system tests created in the system design phase are executed**. **System tests check the complete functionality of the system. In this, more attention is given to performance testing and regression testing.**

- **Acceptance testing:-** the **acceptance tests created in the requirement analysis phase are executed. This testing ensures that the system is compatible with other systems. And in this, non-functional issues like:- load time, performance etc. are tested in the user environment.**

Software Process Model:
Incremental Model



Software Process Model:

Incremental Model

- The **Incremental model** is **split into several iterations** ("**Lego-style**")
- The **Model** consists of **iterative** and **incremental** development stages. The Model is comprised of **several mini waterfall cycles.**
- We **collect the customer's requirements**, now instead of making the entire software at once, we first **take some requirements** and based on them **create a module or function of the software** and **deliver it to the customer**. Then we **take some more requirements** and based on them **add another module to that software with no or little change in earlier to the previous modules, REPEAT till the system is complete**.
- The **development process** can go either **sequentially or in parallel**.
- **Parallel development adds to the speed of delivery**
- The **Model allows software developers to take advantage of learnings** and **insights gleaned from earlier development stages.**

Software Process Model:
Incremental Model

- The **Incremental Model** is a great solution **for projects that need accommodation for some change requests between increments. (*less expensive to change requirements and scope*)**
- This model also yields the benefit of being **able to detect problems earlier** and **Developers** can make **alterations based on the learnings of the previous cycles.**
- Important **modules/functions are developed first and then the rest are added in chunks.**
- This model is **flexible** and The **customer** can **respond** to each **module** and **provide feedback** if any changes are needed. ***Project progress can be measured.***
- A **potential disadvantage** to the Model is the **need for strategic planning and documentation.**
- This method also tends to require **more resources, staff and monetary, behind the project.**

Software Process Model:

Incremental Model: Phases

- **Communication:** we **talk face to face** with the **customer** and **collect** his/her **mandatory requirements**. Like **what functionalities does the customer want in his software**, etc.
- **Planning:** the **requirements** are **divided** into **multiple modules** and **planning** is done on their basis.
- **Modeling:** the **design** of **each module is prepared**. After the design is ready, we **take a particular module** among many modules and save it in DDS (Design Document Specification). Diagrams like ERDs and DFDs are included in this document.

Software Process Model:

Incremental Model: Phases

- **Construction: start construction** based on the **design of that particular module**. That is, the **design of the module is implemented in coding**. **Once the code is written, it is tested.**

- **Deployment: After the testing of the code is completed**, **if the module is working properly then it is given to the customer for use.** After this, **the next module is developed through the same phases** and **is combined with the previous module.** This makes new functionality available to the customer. This will continue until complete modules are developed

Software Process Model:

Incremental Model: Example

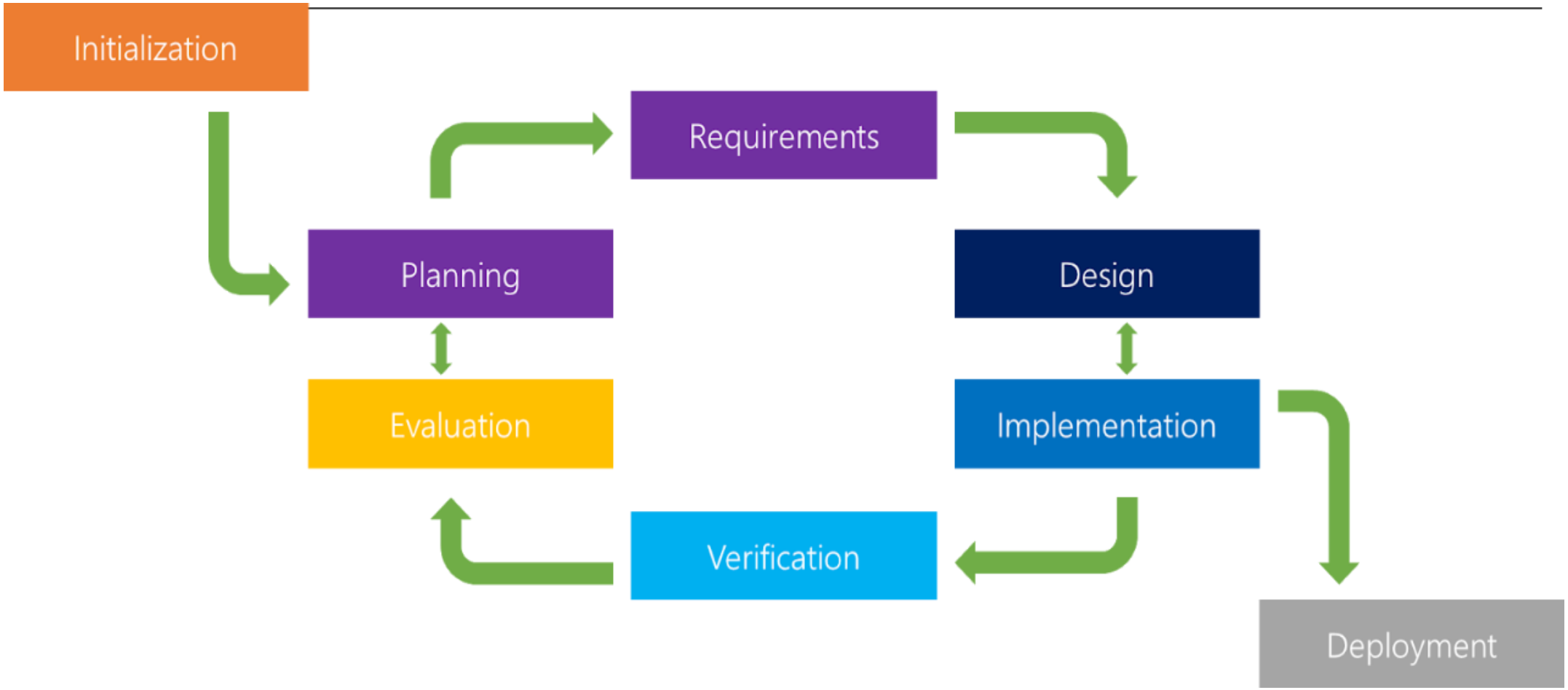
- **How Microsoft Office Suite Uses the Incremental Model: Microsoft Office (Word, Excel, PowerPoint, etc.) was not built all at once but developed incrementally over multiple versions. Each version introduced new features while improving existing ones.**

- **Development Stages:**

1. **Initial Release –** The first version of Microsoft Word, for example, provided only basic text editing features.
2. **First Increment** – Later versions added spell-check, formatting options, and printing capabilities.
3. **Second Increment** – More features like grammar check, templates, and tables were introduced.
4. **Third Increment** – Cloud integration (OneDrive), real-time collaboration, and AI-powered suggestions were added.
5. **Ongoing Development** – Microsoft continues to release updates incrementally, improving security, performance, and features based on user feedback.

Software Process Model:

Iterative Model



Software Process Model:

Iterative Model

- Starts **developing the software with some requirements** and **when it is developed**, it is **reviewed** and **improved in the next iteration and so on.**
- **Iteration means that we are repeating the development process again and again**. We develop the first version of the software following the SDLC process with **some software requirements. We can call this Iteration 1.**
- **After the first version is developed**, **if there is a need to change the software** , **then a new version is developed with the second iteration.** Now again if the new version is not enough, then we will make changes in it with the third iteration. **The iteration will be repeated until the complete software is ready.**
- As **software is delivered in parts**, **there is no need for a full specification** from the project's start.

Software Process Model:

Iterative Model: Phases

1. **Requirement gathering & analysis:** **software requirements** of the customer are **collected** and it is **analyzed** whether those requirements can be meet or not. Besides, it is also **checked** whether this **project will not go beyond budget.**

2. **Design:** the **design of software is prepared.** For this, various diagrams like Data Flow diagram, class diagram, activity diagram, state transition diagram, etc. are used.

3. **Implementation:** Now the **design of software is implemented in coding** through various programming languages.

4. **Testing**: After the coding of the software is done, **it is now tested so that the bugs and errors present in it can be identified**. Testing techniques like **performance testing**, **security testing**, **requirement testing**, **stress testing**, etc. are done.

Software Process Model:

Iterative Model: Phases

5.**Deployment:** Finally, the **software is given to the customer**. **After this the customer starts using that software in his work environment.**

6.**Review:** **After the software is deployed in its work environment**, **it is reviewed**. **If any error/bug is found** or any **new requirements** come in front of developer, then again, these phases are repeated with new iteration and a new version is developed.

7.**Maintenance:** **we look at customer feedback**, **solve problems, fix errors, update software, etc.**

Software Process Model:

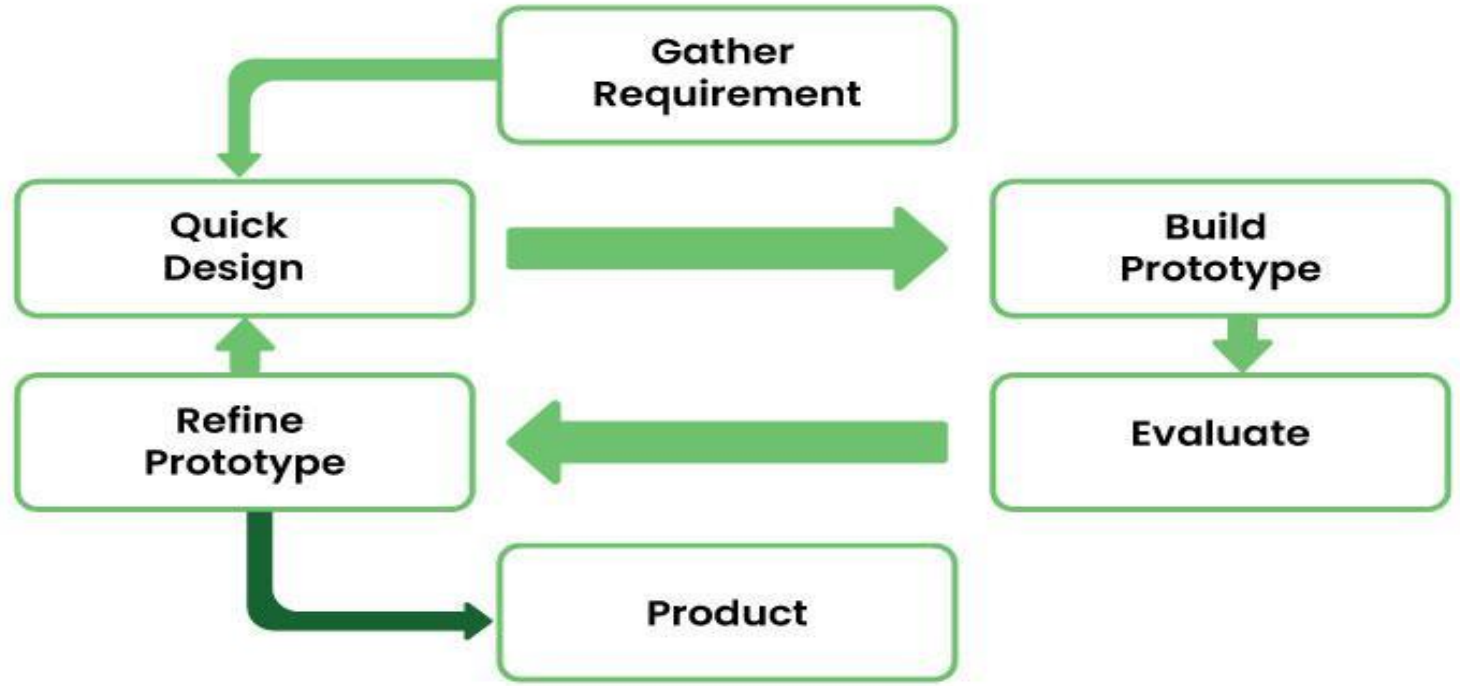
Iterative Model: Example

- **How Google Chrome Uses the Iterative Model:** Google Chrome was not built in a single step; instead, it evolved through multiple iterations, improving its *performance, security, and features over time.*

- **Development Stages (Iterations)**

1. **First Iteration** – Initial release with basic browsing capabilities.
2. **Second Iteration** – Performance improvements, tabbed browsing, and better UI.
3. **Third Iteration** – Security updates, incognito mode, and extension support.
4. **Ongoing Iterations** – Regular updates introduce new features (like AI suggestions), bug fixes, and enhanced security.

Software Process Model:
Prototype Model



Prototype model

Software Process Model:

Prototype Model

- **Prototype model** is an *activity* in which ***instead of freezing the requirements before a design or coding can proceed, a throwaway prototypes of software applications are created (prototype of what is believed the customer wants) which helps to better to understand the requirements. Based on that, final product is manufactured.***
- This prototype is developed based on the ***currently known requirements*** and is created ***when we do not know the requirements well.***
- ***If the end users are not satisfied with the prototype, then a new prototype is created again, consuming more money and time.***

Software Process Model:

Prototype Model: Phases

- **Requirement gathering:** The ***first step*** of prototype model is to ***collect the requirements***, although the ***customer does not know much about the requirements but the major requirements are defined in detail.***
- **Build the initial prototype:** ***the initial prototype is built***. In this some ***basic requirements*** are **displayed** and **user interface** is made available.
- **Review the prototype:** the ***prototype is completed*** is ***presented to the end users or customer*** and ***feedback is taken from them about this prototype.***
- **Revise and improve the prototype:** ***When feedback is taken from end users and customers, the prototype is improved on the basis of feedback***. ***This process continues until the customer gets the prototype as per his desire.***

Software Process Model:

Prototype Model: Example

Microsoft Windows GUI Development : When Microsoft was developing the first versions of Windows (Windows 1.0 and Windows 2.0), they followed the Prototype Model to create and refine the Graphical User Interface (GUI) before finalizing the product.

1. **Why Microsoft Used the Prototype Model?**

- **User Feedback Was Critical –** Since GUIs were a new concept, Microsoft needed user feedback before finalizing the design.
- **Iterative Improvements Were Required** – A static design wouldn't work; continuous refinements were needed.
- **Reducing Development Risks** – A prototype helped Microsoft identify usability issues early on.

Software Process Model:

Prototype Model: Example

2. How the Prototype Model Was Applied?

Phase 1: Initial Requirements Gathering

Microsoft identified key user needs for a GUI-based OS.

They analyzed features from existing systems like **Apple Macintosh** and **Xerox PARC**.

Phase 2: Prototype Development

Engineers built **multiple prototypes** of the Windows GUI.

Early prototypes had **basic windowing, icons, and a file manager**.

These prototypes were tested internally and with selected users.

Software Process Model:

Prototype Model: Example

2. How the Prototype Model Was Applied?

Phase 3: User Evaluation and Feedback

- Users tested the prototypes and provided **feedback** on ***usability, navigation, and design***.
- Microsoft **modified layouts, improved responsiveness, and added new features** like overlapping windows.

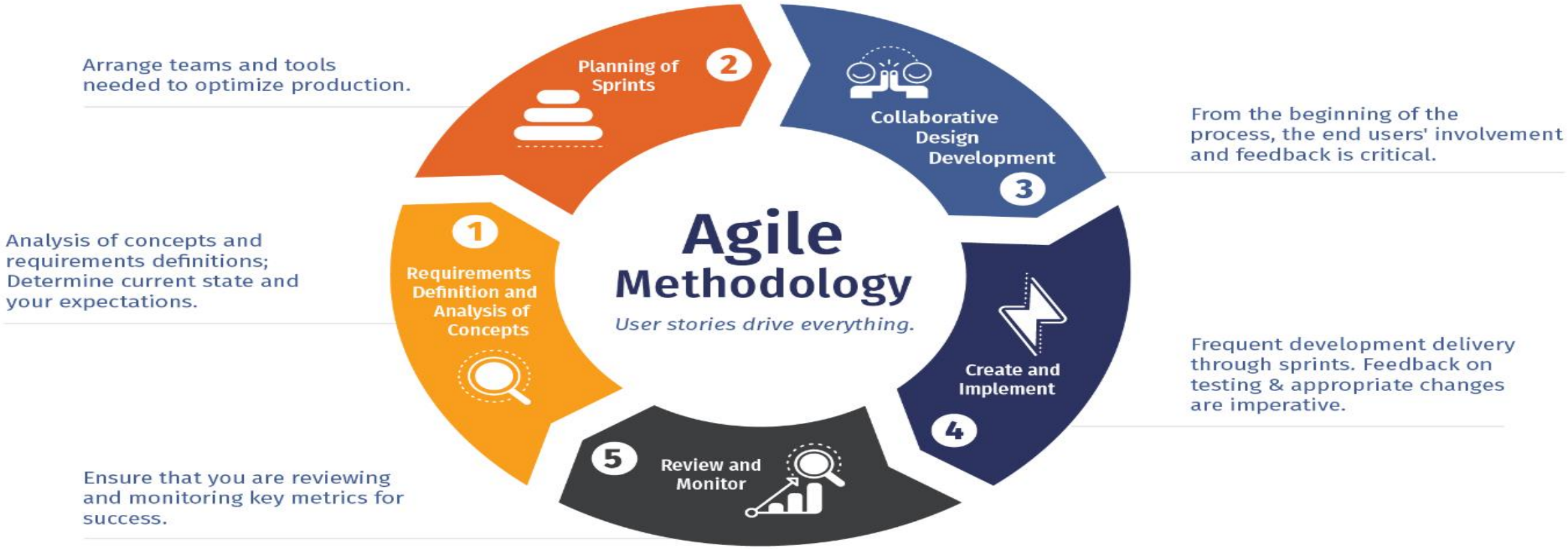
Phase 4: Refinement and Further Iterations

- New prototypes incorporated **user suggestions** (***e.g., better menu structures***).
- The interface became more **intuitive and visually appealing**.

Phase 5: Final Implementation and Deployment

- After multiple iterations, a stable **Windows 1.0 version** was released in **1985**.
- Future versions (Windows 2.0, Windows 3.0) built on the prototype foundation.

Software Process Model:
Agile Group



Software Process Model:

Agile Group

- *Pays less attention to detailed software documentation* (*detailed requirement specification, detailed architecture description*), and *more to software testing activities + process adaptability.*
- **Quick development (***Each iteration typically lasts from about one to three weeks.***)**
- **Maintenance** *more complicated* as ***more time is spent to find the problem when there's no detailed software description.***

Software Process Model:

Agile Group

- At the **heart of Agile** are **iterative** and **Incremental development, intensive communication, process adaptability** and **early customer feedback** ***with rapid delivery of functioning software components.***
- **It breaks the product development into small incremental builds that are then provided in iterations.** The models of this group put more ***focus on delivering a functioning part of the application quickly.***
- Nowadays, **more than 70% of organizations** employ this or that **Agile approach in their IT projects.**

Software Process Model:

Agile Group

- Agile is about **working in close collaboration *both across the team*** and ***with the customers***.

IT IS A MINDSET RATHER THAN A SET OF STRICT PROTOCOLS.

- **Every iteration involves cross functional teams working simultaneously on various areas like: Planning, Requirements, Analysis, Design, Coding, Unit Testing and Acceptance Testing.**
- **At the end of each iteration**, ***stakeholders review the development progress*** and ***re-evaluate the priority of tasks for the future iteration*** to **ensure alignment with user needs.**
- **Agile comes in different flavors.** *The common subtypes are* **Scrum, Extreme Programming, and Kanban.**

Software Process Model:

Agile Group: Manifesto Principles

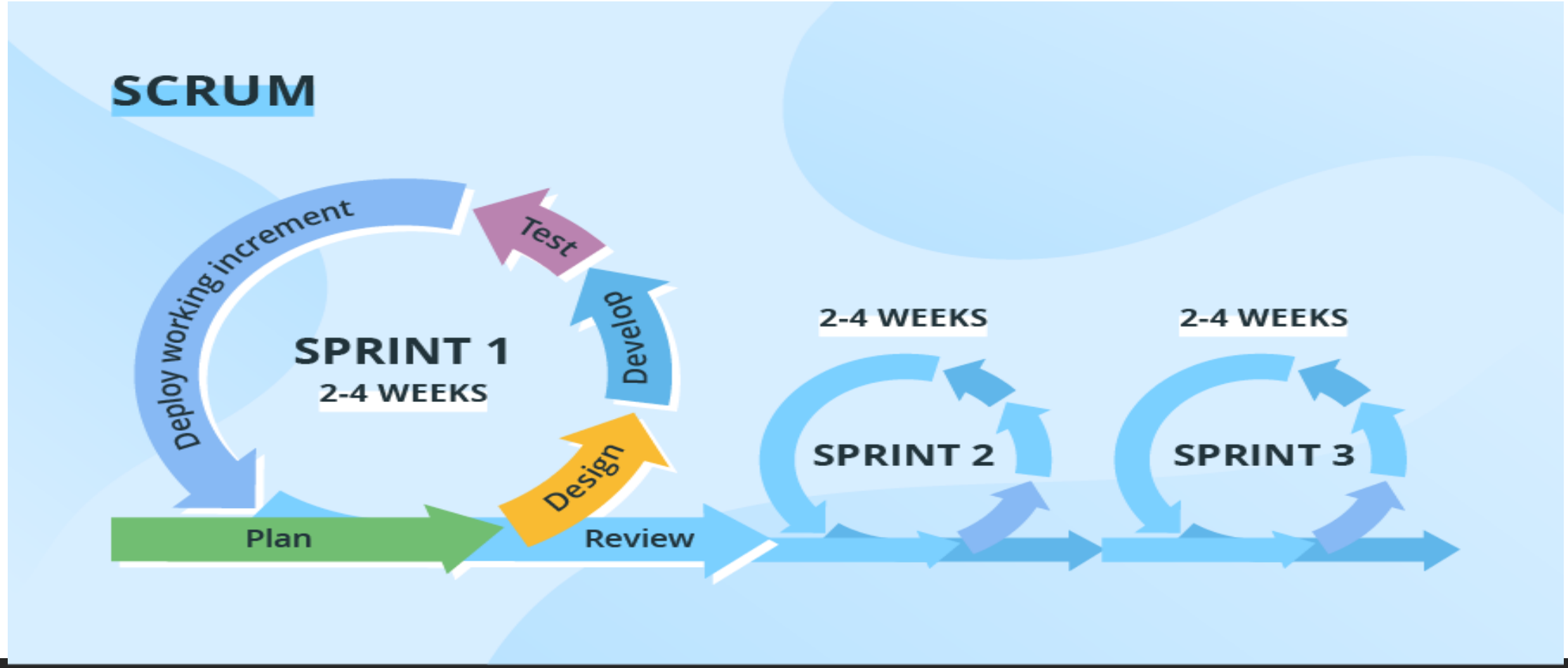
- **Individuals and interactions**: **self-organization** and **motivation** are important, as are **interactions** like **co-location** and **pair programming**.
- **Working software: Demo working software** is considered the **best means of communication** with the **customers to understand their requirements**.
- **Customer collaboration:** requirements are not gathered completely in the beginning of the project. Hence, **continuous customer interaction is very important to get proper product requirements.**
- **Responding to change:** focused on **quick responses to change** and **continuous development.**

Software Process Model:

Agile Group: Manifesto Principles

- **Customer representative in the development team to maintain contact with the customer during software development.** When an iteration is completed, **stakeholders and customer representatives review it and re-evaluate the requirements.**
- **Incremental versions of the software have to be delivered to the customer representative after a few weeks.**
- In this model it is advised that the **size of the development team should be small (5 to 9 people)** so that the **team members can communicate face to face.**
- In agile development, **two programmers work together. One programmer does the coding and the other reviews that code.** Both programmers keep changing their tasks, that is, sometimes one does code and sometimes someone reviews.

Software Process Model:
Agile Group: SCRUM



Software Process Model:
Agile Group: SCRUM

- **Scrum** is the most **popular agile model.** Its **iterations** of software development are known as **sprints.**
- **Scrum** ***is a way to get work done as a team in small pieces at a time***, ***with continuous experimentation*** and ***feedback loops along the way to learn and improve as you go.***
- **During** the **iterations ('sprints'),** which are usually **2–4 weeks long,** teams **assess the previous sprint** and **plan the next sprint**.
- **After each sprint**, **new features/items are added** to be **coded and tested in the next sprint**. *This* ***occurs until all features have been added and the project is deemed ready for release.***

Software Process Model:
Agile Group: SCRUM

- **Scrum** is a ***lightweight framework*** that helps **people, teams and organizations** ***generate value through adaptive solutions for complex problems.***
- ***In a nutshell, 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.***
  3. ***The Scrum Team and its stakeholders inspect the results and adjust for the next Sprint.***
  4. ***Repeat***
- Scrum combines ***events for inspection and adaptation*** within a containing event, **the Sprint**. These events work because they implement the empirical ***Scrum pillars of transparency, inspection, and adaptation.***

Software Process Model:
Agile Group: SCRUM: Pillars

1. *Transparency: The emergent process and work must be visible to those performing the work as well as those receiving the work. With Scrum, important decisions are based on the perceived state of its three formal artifacts. Artifacts that have low transparency can lead to decisions that diminish value and increase risk. Transparency enables inspection. Inspection without transparency is misleading and wasteful.*
2. *Inspection: The Scrum artifacts and the progress toward agreed goals must be inspected frequently and diligently to detect potentially undesirable variances or problems. To help with inspection, Scrum provides cadence in the form of its five events. Inspection enables adaptation. Inspection without adaptation is considered pointless. Scrum events are designed to provoke change.*

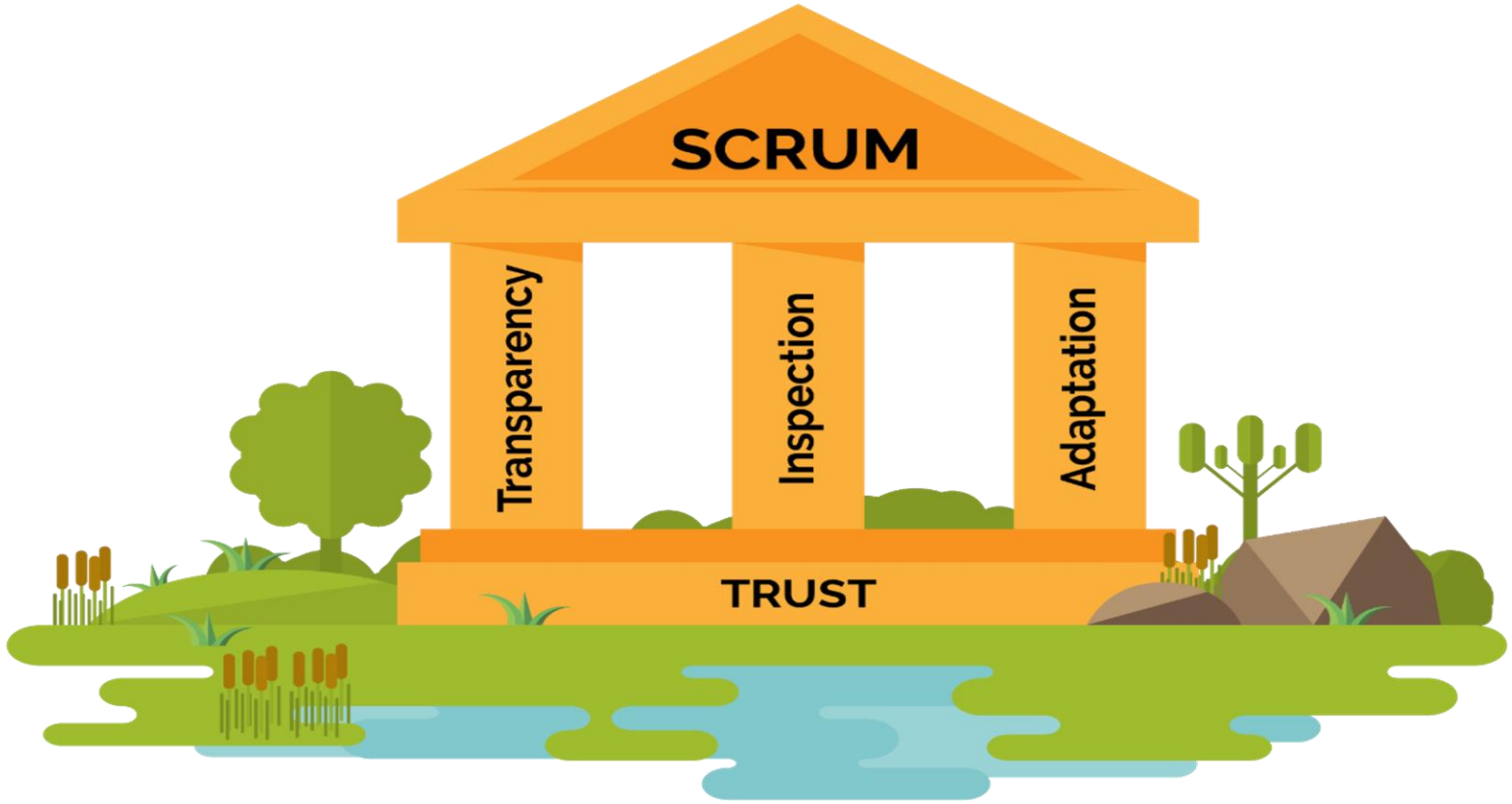
Software Process Model:

Agile Group: SCRUM: Pillars

3. Adaptation: If any aspects of a process deviate outside acceptable limits or if the resulting product is unacceptable, the process being applied or the materials being produced must be adjusted. The adjustment must be made as soon as possible to minimize further deviation.

Adaptation becomes more difficult when the people involved are not empowered or self-managing. A Scrum Team is expected to adapt the moment it learns anything new through inspection.

Software Process Model:
Agile Group: SCRUM: Values



COURAGE
Scrum Team members have courage to do the right thing and work on tough problems

FOCUS
Everyone focuses on the work of the Sprint and the goals of the Scrum Team

COMMITMENT
People personally commit to achieving the goals of the Scrum Team

RESPECT
Scrum Team members respect each other to be capable, independent people

OPENNESS
The Scrum Team and its stakeholders agree to be open about all the work and the challenges with performing the work

Credit: ABN AMRO Bank N.V.

Software Process Model: Agile Group: SCRUM: The Scrum Team

- The fundamental unit of Scrum is a small team of people, a **Scrum Team**. The Scrum Team consists of **one Scrum Master**, **one Product Owner**, and **Developers**. Within a Scrum Team, ***there are no sub-teams or hierarchies.*** It is a cohesive unit of professionals ***focused on one objective at a time, the Product Goal.***
- Scrum Teams are ***cross-functional***, meaning the members ***have all the skills necessary to create value each Sprint***. They are also self-managing, meaning they ***internally decide who does what, when, and how.***
- The Scrum Team is small enough to remain nimble and large enough to complete significant work within a Sprint, typically **10 or fewer people**.
- If ***Scrum Teams become too large, they should consider reorganizing into multiple cohesive Scrum Teams, each focused on the same product***. Therefore, they should **share the same Product Goal, Product Backlog, and Product Owner.**

Software Process Model: Agile Group: SCRUM: The Scrum Team

- The Scrum Team is responsible for all product-related activities from stakeholder collaboration, verification, maintenance, operation, experimentation, research and development, and anything else that might be required.
- The entire Scrum Team is accountable for creating a valuable, useful Increment every Sprint. Scrum defines three specific accountabilities within the Scrum Team: the Developers, the Product Owner, and the Scrum Master.

Software Process Model: Agile Group: SCRUM: The Scrum Team: the Developers

- **Developers** are the people in the ***Scrum Team*** that are **committed to creating any aspect of a usable Increment each Sprint.**
- The ***Developers are always accountable for:***
- **Creating a plan for the Sprint, the Sprint Backlog;**
- **Instilling quality by adhering to a Definition of Done;**
- **Adapting their plan each day toward the Sprint Goal; and,**
- **Holding each other accountable as professionals.**

Software Process Model: Agile Group: SCRUM: The Scrum Team: Product Owner

- **The Product Owner** is accountable for **maximizing the value of the product resulting from the work of the Scrum Team.**
- The Product Owner is also accountable for effective ***Product Backlog management***, which includes:
  - **Developing and explicitly communicating the Product Goal**;
  - **Creating and clearly communicating Product Backlog items**;
  - **Ordering Product Backlog items; and,**
  - **Ensuring that the Product Backlog is transparent, visible and understood.**
- The ***Product Owner is one person, not a committee***. The **Product Owner** may represent the ***needs of many stakeholders*** in the **Product Backlog**. ***Those wanting to change the Product Backlog can do so by trying to convince the Product Owner.***

Software Process Model: Agile Group: SCRUM: The Scrum Team: the Scrum Master

- **The Scrum Master** is accountable for **establishing Scrum as defined in the Scrum Guide** and **for the Scrum Team's effectiveness.** They do this by helping everyone understand Scrum theory and practice, both within the Scrum Team and the organization. Scrum Masters are ***true leaders who serve the Scrum Team and the larger organization.***
- The Scrum Master serves the Scrum Team in several ways, including:
  - **Coaching the team members in *self-management and cross-functionality*;**
  - **Helping the Scrum Team focus on creating high-value Increments that meet the *Definition of Done;***
  - **Causing the *removal of impediments* to the Scrum Team's progress; and,**
  - **Ensuring that all Scrum events take place and are positive, productive, and kept within the timebox.**

Software Process Model: Agile Group: SCRUM: The Scrum Team: the Scrum Master

- The Scrum Master serves the **Product Owner** in several ways, including:
  - **Helping find techniques for *effective Product Goal definition and Product Backlog management;***
  - **Facilitating stakeholder collaboration as requested or needed.**
- The Scrum Master serves the **organization** in several ways, including:
  - **Leading, training, and coaching the organization in its *Scrum adoption*;**
  - **Planning and advising Scrum implementations within the organization;**
  - **Helping employees and stakeholders understand and enact an empirical approach for complex work; and,**
  - **Removing barriers between stakeholders and Scrum Teams.**

Software Process Model:
Agile Group: SCRUM: Events

- The **Scrum Team** takes part in **5 Events** and **produces 3 Artifacts .**
- Each **event** in ***Scrum is a formal opportunity to inspect and adapt Scrum artifacts.*** These events are specifically designed to enable the **transparency** required.
- ***Failure to operate any events as prescribed results in lost opportunities to inspect and adapt.*** Events are used in Scrum to create **regularity** and to ***minimize the need for meetings not defined in Scrum.***
- Optimally**, all events are held at the same time and place to reduce complexity.**

Software Process Model:

Agile Group: SCRUM: Events

1. **The Sprint:** Sprints are the **heartbeat of Scrum**, where **ideas** are **turned** into **value**. It is a ***container for all other events.***

- They are **fixed length events of one month or less to create consistency**. ***A new Sprint starts immediately after the conclusion of the previous Sprint.***
- All the work necessary to achieve the **Product Goal**, including ***Sprint Planning, Daily Scrums, Sprint Review, and Sprint Retrospective***, **happen within Sprints.**

During the Sprint:

- **No changes** are made that would endanger the **Sprint Goal**;
- **Quality** does not decrease;
- The **Product Backlog** is **refined** as needed; and,
- **Scope** may be **clarified** and **renegotiated** with the **Product Owner** as more is learned.

Software Process Model:

Agile Group: SCRUM: Events

1. **The Sprint:**
   - Sprints enable **predictability** by ensuring **inspection** and **adaptation** of progress toward a **Product Goal** ***at least every calendar month.***
   - ***When a Sprint's horizon is too long the*** **Sprint Goal may become invalid; complexity may rise** ***and risk may increase.***
   - **A Sprint could be cancelled** ***if the Sprint Goal becomes obsolete. Only the Product Owner has the authority to cancel the Sprint.***

Software Process Model:

Agile Group: SCRUM: Events

2. Sprint Planning *initiates the Sprint by laying out the work to be performed for the Sprint. This resulting plan is created by the collaborative work of the entire Scrum Team.*

- *The Product Owner ensures that attendees are prepared to discuss the most important Product Backlog items and how they map to the Product Goal.*
- *Sprint Planning addresses the following topics:*
- **Topic One:**
- **Why is this Sprint valuable?** *By Product Owner*
  - **The Product Owner proposes *how the product could increase its value* and utility in the current Sprint. The whole Scrum Team then collaborates to define a Sprint Goal that communicates *why the Sprint is valuable to stakeholders.* The Sprint Goal must be finalized prior to the end of Sprint Planning.**

Software Process Model:

Agile Group: SCRUM: Events

2. Sprint Planning …

- Topic Two:
- What can be Done this Sprint? *By Developer + Product Owner Discussion*
- Through discussion with the Product Owner, the Developers *select items from the Product Backlog to include in the current Sprint.* The Scrum Team may refine these items during this process, which increases understanding and confidence.
- *Selecting how much can be completed within a Sprint may be challenging.* However, the more the Developers know about their past performance, their upcoming capacity, and their Definition of Done, the more confident they will be in their Sprint forecasts.

Software Process Model:

Agile Group: SCRUM: Events

2. Sprint Planning …

- Topic Three:
- How will the chosen work get done? *By Developer*
- *For each selected Product Backlog item, the Developers plan the work necessary to create an Increment that meets the Definition of Done. This is often done by decomposing Product Backlog items into smaller work items of one day or less. How this is done is at the sole discretion of the Developers. No one else tells them how to turn Product Backlog items into Increments of value.*
- *The Sprint Goal, the Product Backlog items selected for the Sprint, plus the plan for delivering them are together referred to as the Sprint Backlog.*
- *Sprint Planning is timeboxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter.*

Software Process Model:

Agile Group: SCRUM: Events

3. The Daily Scrum *is a 15-minute event for the Developers of the Scrum Team to improve communications, identify impediments, promote quick decision-making, and consequently eliminate the need for other meetings.*

- *The purpose of the Daily Scrum is to inspect progress toward the Sprint Goal and adapt the Sprint Backlog as necessary, adjusting the upcoming planned work.*
- *The Developers can select whatever structure and techniques they want, as long as their Daily Scrum focuses on progress toward the Sprint Goal and produces an actionable plan for the next day of work. This creates focus and improves self-management.*

Software Process Model:

Agile Group: SCRUM: Events

4. The Sprint Review *is the second to last event of the Sprint and is timeboxed to a maximum of four hours for a one-month Sprint. For shorter Sprints, the event is usually shorter.*

- *The purpose of the Sprint Review is to inspect the outcome of the Sprint and determine future adaptations. The Scrum Team presents the results of their work to key stakeholders and progress toward the Product Goal is discussed.*
- *During the event, the Scrum Team and stakeholders review what was accomplished in the Sprint and what has changed in their environment.*
- *Based on this information, attendees collaborate on what to do next. The Product Backlog may also be adjusted to meet new opportunities. The Sprint Review is a working session and the Scrum Team should avoid limiting it to a presentation.*

Software Process Model:

Agile Group: SCRUM: Events

5. Sprint Retrospective *is to plan ways to increase quality and effectiveness. The Team inspects how the last Sprint went with regards to individuals, interactions, processes, tools, and their Definition of Done. Inspected elements often vary with the domain of work.*
   - *Assumptions that led them astray are identified and their origins explored. The Scrum Team discusses what went well during the Sprint, what problems it encountered, and how those problems were (or were not) solved.*
   - *The Scrum Team identifies the most helpful changes to improve its effectiveness. The most impactful improvements are addressed as soon as possible. They may even be added to the Sprint Backlog for the next Sprint.*
   - *The Sprint Retrospective concludes the Sprint. It is timeboxed to a maximum of three hours or less for a one-month Sprint.*

Software Process Model:

Agile Group: SCRUM: Artifacts

- Scrum's artifacts represent work or value.
- Each artifact contains a commitment to ensure it provides information that enhances transparency and focus against which progress can be measured:
  - For the Product Backlog it is the Product Goal.
  - For the Sprint Backlog it is the Sprint Goal.
  - For the Increment it is the Definition of Done.
- These commitments exist to reinforce empiricism and the Scrum values for the Scrum Team and their stakeholders.

Software Process Model:

Agile Group: SCRUM: Artifacts

1. **Product Backlog:** is an ***emergent, ordered list of what is needed to improve the product.*** It is the ***single source of work*** undertaken by the Scrum Team.

- ***Product Backlog refinement*** is the act of ***breaking down Product Backlog items into smaller more precise items*** and ***adding details***, such as a ***description***, ***order***, and ***size***.
- The **Developers** who will be doing the work are responsible for the **sizing**. The **Product Owner** may influence the **Developers** by helping them **understand** and **select trade-offs.**

Software Process Model:

Agile Group: SCRUM: Artifacts

1. **Product Backlog: … Commitment: Product Goal**

- **The Product Goal** describes a ***future state of the product which can serve as a target for the Scrum Team to plan against.*** The Product Goal is in the Product Backlog. The rest of the Product Backlog emerges to define "**what**" will fulfill the Product Goal.
  - ***A product is a vehicle to deliver value. It has a clear boundary, known stakeholders, well-defined users or customers. A product could be a service, a physical product, or something more abstract.***
- **The Product Goal** is the **long-term objective for the Scrum Team.** They must fulfill (or abandon) one objective before taking on the next.

Software Process Model:

Agile Group: SCRUM: Artifacts

2.**Sprint Backlog:** is composed of the ***Sprint Goal*** (***why***), ***the set of Product Backlog items selected for the Sprint*** (***what***), as well as an ***actionable plan for delivering the Increment*** (***how***).

- The ***Sprint Backlog is a plan by and for the Developers.*** It is a highly visible, real-time picture of ***the work that the Developers plan to accomplish during the Sprint in order to achieve the Sprint Goal***.
- Consequently, the Sprint Backlog is updated throughout the Sprint as more is learned. It should have enough detail that they can inspect their progress in the Daily Scrum.

Software Process Model:

Agile Group: SCRUM: Artifacts

2.Sprint Backlog: Commitment: Sprint Goal

- **The Sprint Goal** is the **single objective for the Sprint.** Although the Sprint Goal is a commitment by the Developers, it provides flexibility in terms of the exact work needed to achieve it. ***The Sprint Goal also creates coherence and focus, encouraging the Scrum Team to work together rather than on separate initiatives.***
- The **Sprint Goal is created during the Sprint Planning event** and **then added to the Sprint Backlog**. As the Developers work during the Sprint, they keep the Sprint Goal in mind.
- ***If the work turns out to be different than they expected, they collaborate with the Product Owner to negotiate the scope of the Sprint Backlog within the Sprint without affecting the Sprint Goal.***

Software Process Model:

Agile Group: SCRUM: Artifacts

3. **Increment:** is a concrete stepping stone toward the Product Goal. Each Increment is additive to all prior Increments and thoroughly verified, ensuring that all Increments work together. In order to provide value, the Increment must be usable.

- Multiple Increments may be created within a Sprint. ***The sum of the Increments is presented at the Sprint Review thus supporting empiricism.*** However, an Increment may be delivered to stakeholders prior to the end of the Sprint. ***The Sprint Review should never be considered a gate to releasing value.***
- ***Work cannot be considered part of an Increment unless it meets the*** **Definition of Done**.

Software Process Model:
Agile Group: SCRUM: Artifacts

3. Increment: Commitment: Definition of Done

- The **Definition of Done** is a ***formal description*** of the ***state of the Increment when it meets the quality measures required for the product***.
- **The moment a Product Backlog item meets the Definition of Done; an Increment is born.**
- The **Definition of Done** creates **transparency** by providing everyone a **shared understanding of what work was completed as part of the Increment**. ***If a Product Backlog item does not meet the Definition of Done, it cannot be released or even presented at the Sprint Review. Instead, it returns to the Product Backlog for future consideration.***

Software Process Model:

Agile Group: SCRUM: Scrum vs SDLC

- In **Scrum**, the **Software Development Life Cycle (SDLC)** activities such as **requirement elicitation and analysis, design, implementation, and testing** are integrated into various **Scrum events.** ***This approach ensures faster feedback, higher flexibility, and continuous improvement.***

- **Unlike traditional SDLC** models where these activities occur in distinct phases, Scrum promotes an **iterative and incremental** approach, where these activities happen **continuously within each Sprint.**

Software Process Model:

Agile Group: SCRUM: Scrum vs SDLC

- **Mapping SDLC Activities to Scrum Events:**

1. **Requirement Elicitation and Analysis in Scrum:**

   1. **Product Backlog Refinement (Grooming):**

   This is a **continuous activity** where the **Product Owner (PO)** works with the Scrum Team to break down high-level ideas into detailed **User Stories** with **clear acceptance criteria.** ***The team asks questions, identifies dependencies, and estimates the effort required.***

   Key Participants: Product Owner, Scrum Team, sometimes stakeholders.

   Outcome: Well-defined, prioritized Product Backlog Items (PBIs).

   2. **Sprint Planning:**

   At the start of each Sprint, the team selects items from the Product Backlog and further analyzes them to understand **what needs to be built.** This involves detailed discussions to ***clarify requirements, define the Sprint Goal, and plan the work.***

   Outcome: A clear understanding of the ***Sprint Scope*** and the ***work needed to achieve the Sprint Goal.***

Software Process Model:

Agile Group: SCRUM: Scrum vs SDLC

- **Mapping SDLC Activities to Scrum Events:**

2. Design in Scrum:

1. Sprint Planning:

After clarifying the requirements, the team discusses **how** to implement the features. This includes **high-level architecture decisions**, defining technical tasks, and identifying potential risks.

- **Example:** ***For a new API feature, the team might outline the endpoints, data models, and authentication mechanisms.***

2. Daily Scrum (Stand-up):

Design discussions don't stop after Sprint Planning. During daily stand-ups, team members may identify technical blockers that require quick design decisions or adjustments.

- **Example:** ***A developer facing an integration issue may propose a new design approach to the team.***

3. Ad-hoc Design Sessions:

Scrum allows flexibility for spontaneous discussions. If a complex design problem arises, the team can have ***impromptu whiteboard sessions*** without waiting for the next Sprint Planning.

Software Process Model:

Agile Group: SCRUM: Scrum vs SDLC

- **Mapping SDLC Activities to Scrum Events:**

3. Implementation (Development) in Scrum:

1. Sprint Execution:

The actual coding happens here. Developers work on tasks defined during Sprint Planning, following the **Definition of Done (DoD),** which includes ***coding standards, code reviews, and testing requirements.***

- ***Pair Programming*** and ***code reviews*** are common Agile practices during this phase to ensure quality.
- Developers also ***integrate code regularly (Continuous Integration)*** to detect issues early.

2. Daily Scrum:

A 15-minute time-boxed event to discuss:

- ***What was done yesterday?***
- ***What will be done today?***
- ***Are there any blockers?***

This helps maintain transparency and alignment.

Software Process Model:

Agile Group: SCRUM: Scrum vs SDLC

- **Mapping SDLC Activities to Scrum Events:**

4. Testing (Verification & Validation) in Scrum:

1. **Integrated Testing During Sprint:**

In Scrum, testing is continuous. **Developers** and **QA engineers** collaborate closely. Testing includes:

- **Unit Testing:** ***Written by developers.***
- **Integration Testing:** ***Ensures modules work together.***
- **Automated Testing:** ***Often integrated into CI/CD pipelines.***
- **Exploratory Testing:** ***Done manually to find unexpected issues.***

Software Process Model:

Agile Group: SCRUM: Scrum vs SDLC

- **Mapping SDLC Activities to Scrum Events:**

4. Testing (Verification & Validation) in Scrum:

2. Sprint Review:

The team demonstrates the working product to stakeholders. This serves as an opportunity to ***validate if the increment meets the acceptance criteria and business needs.***

- **Feedback Loop:** ***Stakeholders provide feedback, which can lead to new backlog items.***

3. Sprint Retrospective:

After the Sprint, the team reflects on what went well, what didn't, and how to improve. Issues related to testing efficiency, automation gaps, or defect management are discussed here.

- **Outcome:** ***Action items for continuous improvement in the next Sprint.***

Software Process Model:

Agile Group: SCRUM: Scrum vs SDLC

- **Key Differences from Traditional SDLC:**

Traditional SDLC	Scrum (Agile SDLC)
Sequential phases (Waterfall model)	Iterative and incremental development
Separate phases for design, coding, testing	Continuous activities within each Sprint
Late feedback from stakeholders	Early and continuous feedback (Sprint Reviews)
Testing happens after development	Testing happens concurrently with development

Software Process Model:

Agile Group: SCRUM: Example

- A well-known real-world example of software developed using the **Scrum model** is **Spotify**. Spotify, the popular music streaming service.

- **How Spotify Uses Scrum:**

1. **Squads**: Spotify organizes its teams into small, cross-functional groups called "**squads**," which operate like Scrum teams. Each squad focuses on a specific feature or component of the product.
2. **Sprints**: Squads work in time-boxed iterations (sprints), typically lasting **2-4 weeks**, to deliver incremental improvements to the software.
3. **Product Owner**: Each squad has a **Product Owner** who prioritizes the **backlog** and ensures the team is working on the most valuable features.
4. **Scrum Master**: A Scrum Master **facilitates the process**, **removes impediments**, and **ensures the team adheres to Scrum practices.**
5. **Continuous Delivery**: Spotify emphasizes **frequent releases and continuous integration**, allowing them to quickly deliver new features and updates to users.

Software Process Model:
Agile Group: SCRUM: Example

1. **Requirement Elicitation and Analysis:**
    1. **Scrum Framework**: In Scrum, requirements are captured in the **Product Backlog**, which is a prioritized list of features, user stories, and tasks.
    2. **Spotify Example**:
        - The ***Product Owner*** works with stakeholders (e.g., users, business teams, and designers) to ***gather requirements*** for new features, such as ***personalized playlists or improved search functionality***.
        - ***User stories*** are ***written and refined*** during **Backlog Refinement** sessions. For example: ***"As a user, I want to discover new music based on my listening history so that I can explore new artists."***
        - The team analyzes these requirements to ensure they are clear, feasible, and aligned with business goals.

Software Process Model:
Agile Group: SCRUM: Example

2. **Design:**

1. **Scrum Framework**: During **Sprint Planning**, the team ***selects items from the Product Backlog*** and ***breaks them into tasks.*** Design discussions happen as part of this process.

2. **Spotify Example**:

- The **squad (team)** collaborates to design the technical and user experience (UX) aspects of a feature. For example, ***designing the algorithm for personalized playlists or the UI for a new search interface.***
- Designers, developers, and architects work together to create wireframes, system diagrams, and prototypes.
- The output is a clear plan for implementation, which is added to the ***Sprint Backlog.***

Software Process Model:
Agile Group: SCRUM: Example

3. Implementation (Development):

1. **Scrum Framework**: The development team works on the tasks in the Sprint Backlog during the sprint, which typically lasts **2-4 weeks.**
2. **Spotify Example:**
   - **Developers** write **code to implement the feature**, such as ***building the backend logic for playlist recommendations or integrating the new search functionality.***
   - **Continuous integration (CI)** tools are used to merge code into the main branch frequently, ensuring that the software is always in a releasable state.
   - **Pair programming, code reviews**, and **collaboration** are common practices to ensure high-quality code.

Software Process Model:
Agile Group: SCRUM: Example

4. **Testing:**

   1. **Scrum Framework**: Testing is integrated throughout the sprint, not as a separate phase. ***The team writes automated tests*** (**unit, integration, regression tests** and **manual testing**)

   2. **Spotify Example:**

      - **Testers** (***or developers in a cross-functional team***) write **automated tests** for the new feature, such as ***testing the accuracy of playlist recommendations or the responsiveness of the search feature.***
      - **QA engineers** perform ***exploratory testing to identify edge cases or usability issues.***
      - ***Bugs are logged and fixed within the same sprint***, ensuring that the feature is shippable by the end of the iteration.

Software Process Model:

Agile Group: SCRUM: Example

5. Deployment and Feedback:

1. **Scrum Framework**: At the end of each sprint, the team delivers a potentially shippable product increment. This is reviewed in the **Sprint Review** meeting, where stakeholders provide feedback.
2. **Spotify Example:**
   - The new feature (***e.g., personalized playlists)*** is deployed to a subset of users to gather real-world feedback.
   - **Metrics**, such as ***user engagement or error rates***, are monitored to assess the feature's performance.
   - **Feedback** from users and stakeholders is used to refine the feature in subsequent sprints.

Software Process Model:

Agile Group: SCRUM: Example

6. **Maintenance and Continuous Improvement:**

1. **Scrum Framework**: After deployment, the team continues to monitor and improve the feature based on user feedback and performance data. This is part of the **continuous improvement** philosophy in Scrum.

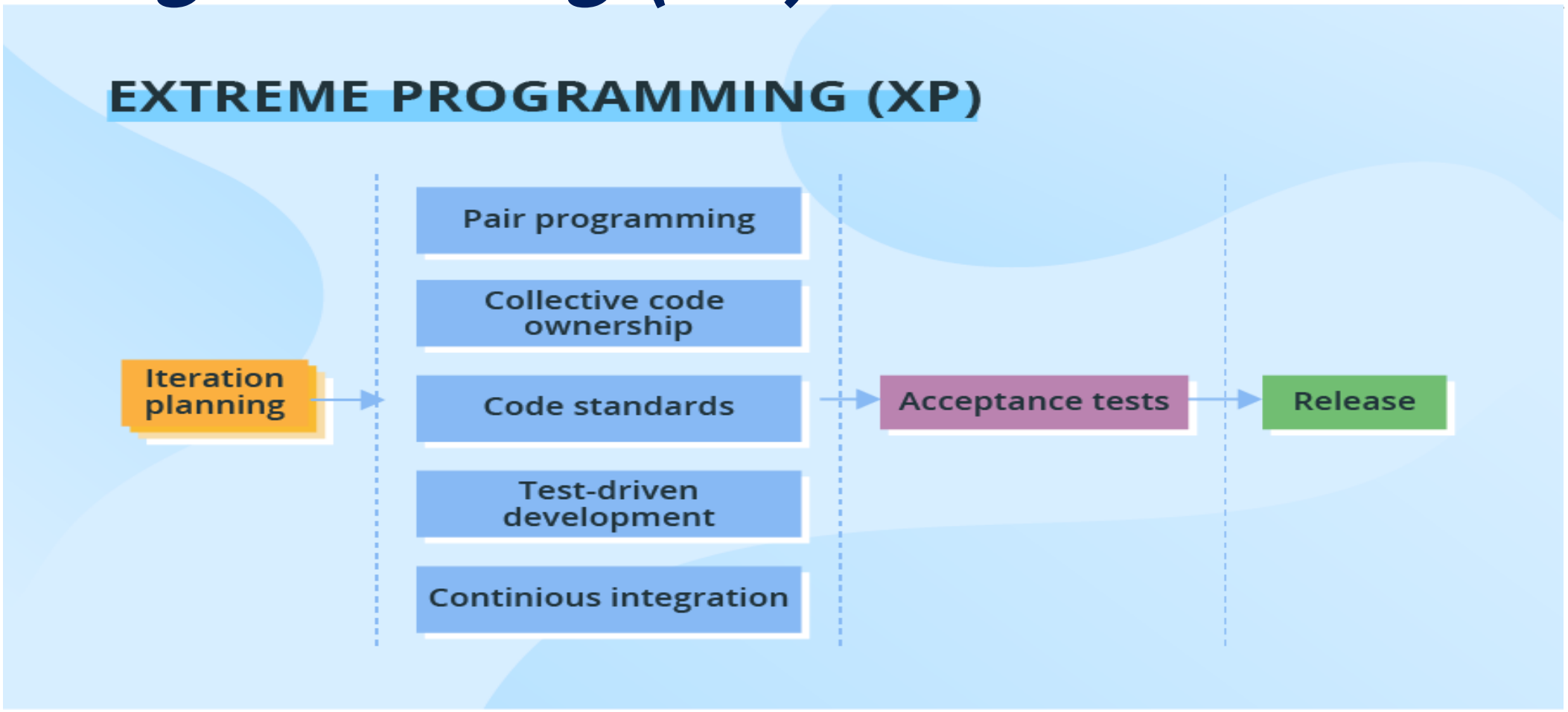
2. **Spotify Example:**

- ***If users report issues with the playlist recommendations, the squad addresses them in the next sprint***.
- The team also looks for opportunities to optimize the feature, such as ***improving the algorithm or reducing latency.***

Software Process Model:
Agile Group: SCRUM: Example

- **How Scrum Iterates Through SDLC:**
  - In Scrum, the SDLC phases are not one-time activities but are repeated in every sprint. For example:
    1. **Sprint 1:** ***Elicit requirements, design, implement, and test a basic version of the playlist feature.***
    2. **Sprint 2:** ***Refine requirements, improve the design, add new functionality, and test again.***
    3. **Sprint 3:** ***Optimize the feature based on user feedback and fix any bugs.***
- **This iterative approach allows Spotify to deliver value incrementally and adapt to changing user needs.**

Software Process Model:
Agile Group: Extreme Programming (XP)



Software Process Model: Agile Group: Extreme Programming (XP)

- **Extreme programming (XP) Model** that ***targets*** **speed** and **simplicity** with ***short development cycles(typical iteration lasts 1-2 weeks)*** and **less documentation.**
- ***Just like other models,*** **XP** ***responds to customer`s stories***, ***adapt & change in real-time***, ***test-driven*** development ***but unlike other methods***, **XP is** ***more disciplined*** and has **strict rules** and **guiding values** that govern how the work gets done.
- The model ***allows changes*** to be introduced ***even after the iteration's launch if the team hasn't started to work with the relevant software piece yet***. Such ***flexibility significantly complicates the delivery of quality software.***

Software Process Model: Agile Group: Extreme Programming (XP)

- The **process structure** is determined by **five guiding values**, **five rules**, and **12 XP practices**
- XP is a **lightweight**, **efficient**, **low-risk**, **flexible**, **predictable**, and **scientific** way to develop a software
- XP was conceived and developed to ***address the specific needs of software development by small teams in the face of vague and changing requirements.***

Software Process Model: Agile Group: Extreme Programming (XP): Lifecycle

- The **XP lifecycle** encourages ***continuous integration where the team member integrates almost constantly***, **as frequently as hourly or daily**.
  - **Pull unfinished work from user stories**
  - **You prioritize the most important items**
  - **Begin iterative planning**
  - **Incorporate honest planning**
  - **Stay in constant communication with all stakeholders and empower the team**
  - **Release work**
  - **Receive feedback**
  - **Return to the iterative planning stage and repeat as needed.**

Software Process Model: Agile Group: Extreme Programming (XP): 5 Values

1. **Simplicity:** *Before starting any XP work, first ask yourself: What is the simplest thing that also works?* The "that works" *part is a key differentiator. In XP, your focus is on getting the most important work done first. This means you're looking for a simple project that you know you can accomplish.*
2. **Communication:** XP relies on *quick reactivity* and *effective communication*. In order to work, the *team needs to be open* and *honest with one another*. *When problems arise, you're expected to speak up. The reason for this is that other team members will often already have a solution. And if they don't, you'll come up with one faster as a group than you would alone.*

Software Process Model: Agile Group: Extreme Programming (XP): 5 Values

3. **Feedback:** XP incorporates ***user stories*** and ***feedback directly into the process.*** ***XP's focus is producing work quickly and simply, then sharing it to get almost immediate feedback.*** In XP, you ***launch frequent releases to gain insights early and often***.

When you receive feedback, you'll adapt the process to incorporate it (instead of the project). For example, if the feedback relieves unnecessary lag time, you'd adjust your process to have a pair of developers improve lag time instead of adjusting the project as a whole.

Software Process Model: Agile Group: Extreme Programming (XP): 5 Values

4. **Courage:** You're always *expected to give honest updates on your progress*, *If you miss a deadline in XP*, *your* **team lead** *likely won't want to discuss why.* ***Instead, you'd tell them you missed the deadline, hold yourself accountable, and get back to work.***

If you're a **team lead**, ***your responsibility at the beginning of the XP process*** is to ***set the expectation for success*** and ***define "done."*** *There is often little planning for failure because the team focuses on success. However, this can be scary, because things won't always go as planned. But if things change during the XP process, your team is expected to adapt and change with it.*

Software Process Model:
Agile Group: Extreme Programming (XP): 5 Values

5. Respect: *In order for teams to communicate and collaborate effectively, they need to be able to disagree. But there are ways to do that kindly.* Respect *is a good foundation that leads to* kindness *and* trust *—even in the presence of a whole lot of honesty.*

For extreme programming, the expectations are:

A. Mutual respect *between customers and the development team.*

B. Mutual respect *between team members.*

C. A recognition that *everyone on the team brings something valuable to the project.*

Software Process Model: Agile Group: Extreme Programming (XP): 5 Rules

1. **Planning:** Determining if the project is **viable** and **the best fit for XP**. ***To do this, you'll look at:***

- **User stories** to see if they match the **simplicity value** and check in to ensure that the customer is available for the process. ***If the user story is more complex, or it's made by an anonymous customer, it likely won't work for XP.***
- The **business value** and **priority of the project** ***to make sure that this falls in line with "getting the most important work done first."***

Software Process Model: Agile Group: Extreme Programming (XP): 5 Rules

1. Planning: …

- *Once you've confirmed the project is viable for XP, create a release schedule—but keep in mind that you should be releasing early and often to gain feedback. To do this:*
  - *Break the project down into* **iterations** and *create a* **plan** *for each one.*
  - *Set* **realistic deadlines** and *a sustainable pace.*
  - *Share real-time updates that help the team identify, adapt, and make changes more quickly.*
  - *Use a project management tool* to create a *Kanban board* or *timeline to track your progress in real-time.*

Software Process Model: Agile Group: Extreme Programming (XP): 5 Rules

2. **Managing:** One of the key elements of XP is the **physical space**. *XP purists recommend using an open workspace where all team members work in one open room. You'll benefit from having a space where you can physically come together. If you work on a remote team, use a platform that encourages asynchronous work for remote collaboration.*

   Daily standups meetings *to check-in* and *encourage constant, open communication. Use both a* **weekly cycle** *and* **quarterly cycle**. *During* **quarterly cycle**, *you and your team will* **review stories** *that will guide your work. You'll look for gaps or opportunities to make changes. Then, you'll work in* **weekly cycles**, *which each start with a customer meeting. The customer chooses the user story they want programmers to work on that week.*

Software Process Model: Agile Group: Extreme Programming (XP): 5 Rules

2. **Managing:** …

- As a **manager or team lead,** your focus will be on *maintaining work progress*, *measuring the pace*, *shifting team members around to address bugs or issues as they arise*, or *changing the XP process to fit your current project and iteration*. ***Remember, the goal of XP is to be* flexible *and* take action*, *so your work will be highly focused on the* team's current work *and* reactive to any changes.**

Software Process Model: Agile Group: Extreme Programming (XP): 5 Rules

3. **Designing:** XP *begin with the simplest possible design*, *knowing that later iterations will make them more complex.* ***Do not add functionality at this stage early; keep it as bare bones as possible.***

Teams often use ***class-responsibility-collaboration (CRC) cards to show how each object in the design interacts.*** ***By filling out each field in the card, you'll get a visual interaction of all the functions as they relate and interact.*** **CRC cards include:**

1. *Class (collection of similar objects)*
2. *Responsibilities (related to the class)*
3. *Collaborators (class that interacts with this one)*

Software Process Model: Agile Group: Extreme Programming (XP): 5 Rules

4. **Coding:** One of the more **unique aspects of XP** is that you'll ***stay in constant contact with the customer throughout the coding process.*** *This partnership allows you to test and incorporate feedback within each iteration, instead of waiting until the end of a sprint. But coding rules are fairly strict in XP.* **Some of these rules include: …**

Software Process Model: Agile Group: Extreme Programming (XP): 5 Rules

4. **Coding: … Some of these rules include:**
    1. All code must meet *coding standards*.
    2. Using a *unit test to nail down requirements and develop all aspects of the project.*
    3. *Programming as a pair—two developers work together simultaneously on the same computer to produce the highest quality results.*
    4. *Use continuous integrations to add new code and immediately test it.*
    5. *Only one pair can update code at any given time to reduce errors.*
    6. *Collective code ownership—any member of the team can change your code at any time.*

Software Process Model: Agile Group: Extreme Programming (XP): 5 Rules

5. Testing: You should be testing throughout the XP process. *All code will need to pass unit tests before it's released. If you discover bugs during these tests, you'll create new, additional tests to fix them. Later on, you'll configure the same* user story *you've been working on into an acceptance test. During this test, the customer reviews the results to see how well you* translated the user story into the product.

Software Process Model: Agile Group: Extreme Programming (XP): 12 Practices

1. The Planning Game: *XP planning is used to guide the work. The results of planning should be what you're hoping to accomplish and by when, and what you'll do next.*
2. Customer Tests: *When you finish a new feature, the customer will develop an acceptance test to determine how close it is to their original user story.*
3. Small Releases: *XP uses small, routine releases to gain insights throughout the process. Often, releases go straight to the customers.*
4. Simple Design: *The XP system is designed for simplicity—you'll produce only what is necessary and nothing more.*

Software Process Model: Agile Group: Extreme Programming (XP): 12 Practices

5.Pair Programming: *All programming comes from a pair of developers who sit side by side. There is no solo work in XP.*

6.Test-Driven Development (TDD): *XP's reliance on feedback requires heavy testing. Through short cycles, programmers release automated tests and then immediately react.*

7.Refactoring: *This is where you'll pay attention to the finer details of the codebase*, ***removing duplicates*** and ***making sure that the code is cohesive.*** *This results in good, simple designs.*

8.Collective Ownership: *Any coding pair can change the code at any time, whether or not they developed it.* ***XP produces code as a team, and everyone's work is held to the higher collective standards.***

Software Process Model: Agile Group: Extreme Programming (XP): 12 Practices

9. **Continuous Integration:** *XP teams don't wait for completed iterations, they integrate constantly. Often, an XP team will integrate multiple times a day.*

10. **Sustainable Pace:** *The intensity of XP works requires, teams should decide how much work they can produce in this way per day and per week, and use that to set work deadlines.*

11. **Metaphor:** The metaphor is, quite literally, a metaphor. *It's decided as a team, and uses language to describe how the team should function. For example, we're ants working as a collective to build up the anthill.*

12. **Coding Standard:** *XP teams follow one standard. XP developers code in the same, unified way so that it reads like one developer.*

Software Process Model: Agile Group: Extreme Programming (XP): Vs. SDLC

- **Extreme Programming (XP)** follows the Software Development Life Cycle (SDLC) activities but in an ***iterative and incremental manner.*** Below is how XP aligns with traditional SDLC activities like **requirement elicitation and analysis**, **design**, **implementation**, and **testing** in a real-world software development scenario:

1. **Requirement Elicitation and Analysis**
   - **Traditional SDLC**: Requirements are gathered upfront in a detailed document (e.g., SRS).
   - **XP Approach**:
     - **User Stories**: Requirements are captured as user stories, which are short, simple descriptions of features from the perspective of the end-user.
     - **On-site Customer**: A customer or product owner is actively involved in the team to clarify requirements and prioritize user stories.
     - **Iterative Planning**: Requirements are not fixed; they evolve over time through frequent feedback loops.
     - **Example**: In a banking app, a user story might be: "***As a user, I want to transfer money to another account so that I can send payments***."

Software Process Model: Agile Group: Extreme Programming (XP): Vs. SDLC

2. **Design**
   - **Traditional SDLC**: A detailed design document is created before implementation.
   - **XP Approach**:
     - **Simple Design:** The team focuses on creating the simplest design that works for the current iteration. Over-engineering is avoided.
     - **Refactoring:** The design evolves continuously through refactoring to improve code quality and adapt to changing requirements.
     - **Pair Programming:** Two developers work together on the same code, which helps in real-time design discussions and knowledge sharing.
     - **Example:** For the banking app, ***the team might design a simple money transfer feature with basic validation rules, which can be enhanced later.***

Software Process Model: Agile Group: Extreme Programming (XP): Vs. SDLC

3. **Implementation**
   - **Traditional SDLC**: Implementation happens after the design phase, often in a linear fashion.
   - **XP Approach**:
     - **Small Releases:** Features are implemented in small, incremental releases, allowing for frequent delivery of working software.
     - **Coding Standards:** The team follows strict coding standards to ensure consistency and readability.
     - **Continuous Integration:** Code is integrated into the main branch frequently (multiple times a day), and automated builds are run to detect integration issues early.
     - **Example:** ***The money transfer feature is implemented in small steps, such as validating account numbers, checking balances, and processing transactions.***

Software Process Model: Agile Group: Extreme Programming (XP): Vs. SDLC

4. **Testing**
   - **Traditional SDLC**: Testing is often a separate phase after implementation.
   - **XP Approach**:
     - **Test-Driven Development (TDD):** Developers write automated unit tests before writing the actual code. The code is then written to pass these tests.
     - **Continuous Testing:** Tests are run continuously during development to ensure immediate feedback on code quality.
     - **Acceptance Testing:** The customer defines acceptance tests for user stories, which are automated and run frequently to ensure the software meets requirements.
     - **Example:** ***For the money transfer feature, unit tests are written to validate account numbers, and acceptance tests ensure the feature works as expected from the user's perspective.***

Software Process Model: Agile Group: Extreme Programming (XP): Example

- **Real-World Example: Developing an E-Commerce Platform Using XP**

1.Requirement Elicitation:

- **User stories** are created for features ***like product search, shopping cart, and checkout.***
- The customer prioritizes the **shopping cart feature for the first iteration**.

2. **Design:**

- A **simple design for the shopping cart is created**, focusing on ***adding/removing items*** and ***calculating totals.***
- **Pair programming** is used to brainstorm and implement the design.

Software Process Model: Agile Group: Extreme Programming (XP): Example

- **Real-World Example: Developing an E-Commerce Platform Using XP**

3. **Implementation:**
   - The ***shopping cart feature is implemented in small increments***, with continuous integration ensuring the code is always deployable.
   - Coding standards are followed to maintain code quality.
4. **Testing:**
   - ***Unit tests are written for functions like adding items and calculating totals.***
   - Acceptance tests ensure the shopping cart works as expected from the user's perspective.

Software Process Model: Agile Group: Extreme Programming (XP): When?

- Manage a **smaller team**. Because of its highly collaborative nature, ***XP works best on smaller teams of under 10 people.***
- Are in **constant contact with your customers**. XP incorporates ***customer requirements throughout the development process***, and even **relies on them** for ***testing*** and ***approval***.
- Have an **adaptable team that can embrace change *without hard feelings***. **XP** will often ***require the whole team to toss out their hard work***. ***There are also rules that allow other team members to make changes at any time, which doesn't work if your team members might take that personally.***
- **Are well versed in the technical aspects of coding**. **XP isn't for beginners**. ***You need to be able to work and make changes quickly.***

Object Oriented System Development Methodology

- The advent of the *object-oriented paradigm* brought ***data*** and ***methods*** together into a ***reusable blueprint***. ***At the beginning of every new object is a*** **class** that defines the ***attributes*** (***variables***) and ***behaviors*** (***methods***) of that ***object***.
- In ***Object Oriented approach***, the **data** and the **operations** ***are present in*** **objects** as ***equal partners.*** **Data** and **operations** are considered of the ***same importance***; ***neither takes precedence over the other.***
- Object-oriented tackle ***large problems*** by ***breaking them into smaller***, more ***manageable chunks*** and ***assign each part of a project to a team for development***, ***which is very difficult to manage in procedural development avoiding the DOMINO EFFECT in procedural development.***

Object Oriented System Development Methodology

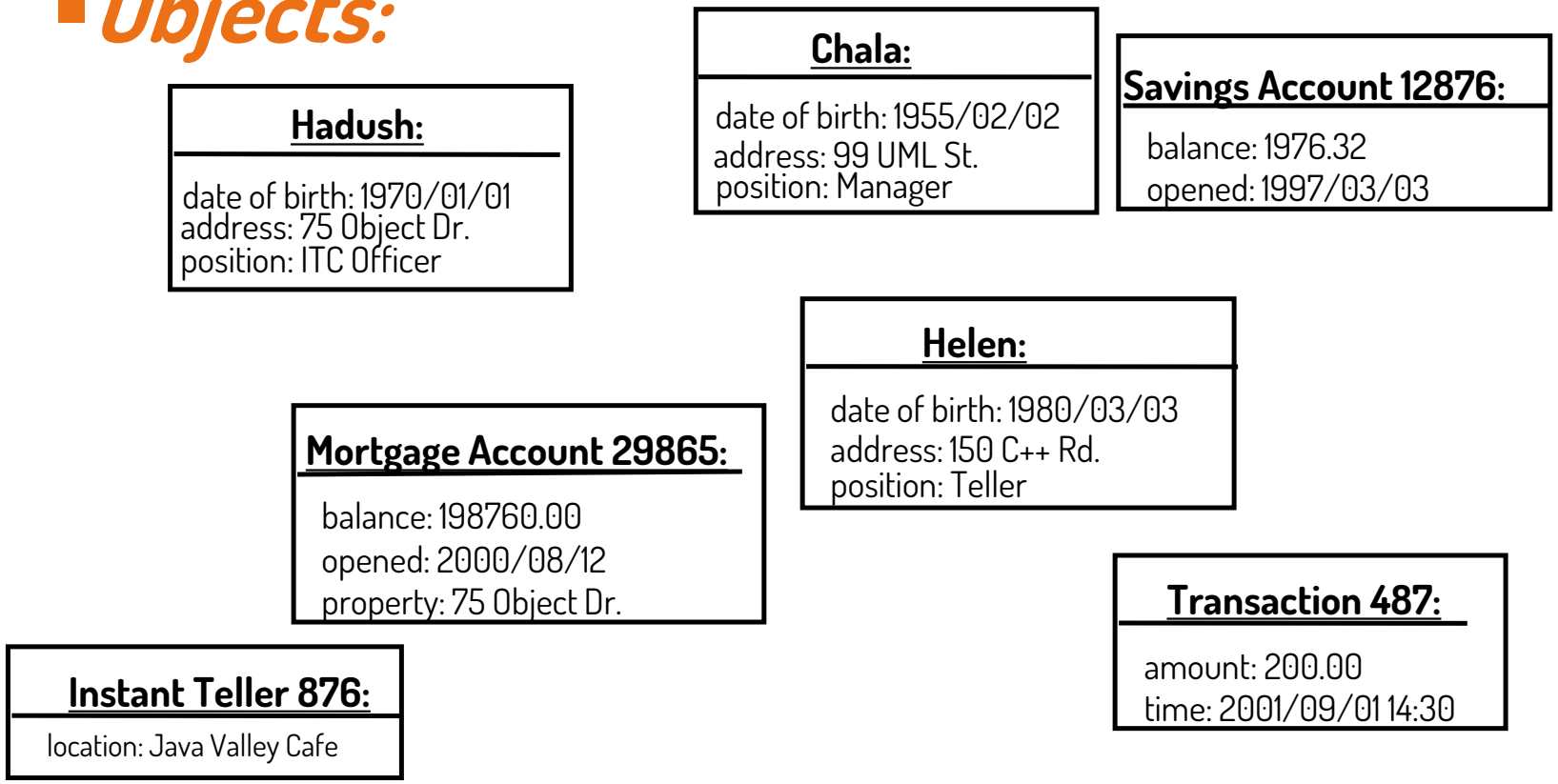
- The *object model encompasses the principles of* abstraction, encapsulation, modularity, hierarchy, typing, concurrency, and persistence.
- OO is an *approach to the solution of problems in which all computations are performed in the context of* objects.
- A *running program* can be seen as a *collection of objects collaborating to perform a given task*
- *Implementation details are hidden; the only communication with an object is* via messages *sent to that object.*

Object Oriented System Development Methodology

- *Objects: Anything that we can describe in the real world/physical or conceptual.* Has properties: (*Represent its state*). Has behaviors(*How it acts and reacts*)
- According to Tsang, "*an object is a self-contained entity with well-defined characteristics (attributes) and behaviors*", *for example, the real-life environment consists of objects such as schools, students, teachers and courses which are related in one way or another.*
- According to Charles Peirce, *"an object is anything that we can think, i.e. anything we can talk about."*
- According to Booch, *"you can do things to the object, and it can do things to other objects, as well"*

Object Oriented System Development Methodology

- *Objects:*



Object Oriented System Development Methodology

- **A class:** *defines how an object is to be built, referred as a blueprint. "No architect would dare build a building or other object without first having a design in place."*
- *Using the class blueprint (unit of abstraction representing similar objects), we can build as many objects as needed.*
- *The things built (created) from the blueprint are objects. A specific object is known as an instance.*

"a class is the concept behind an object, or the "essence" of an object, while an object itself is a tangible entity with a place in space and time (or system memory)"

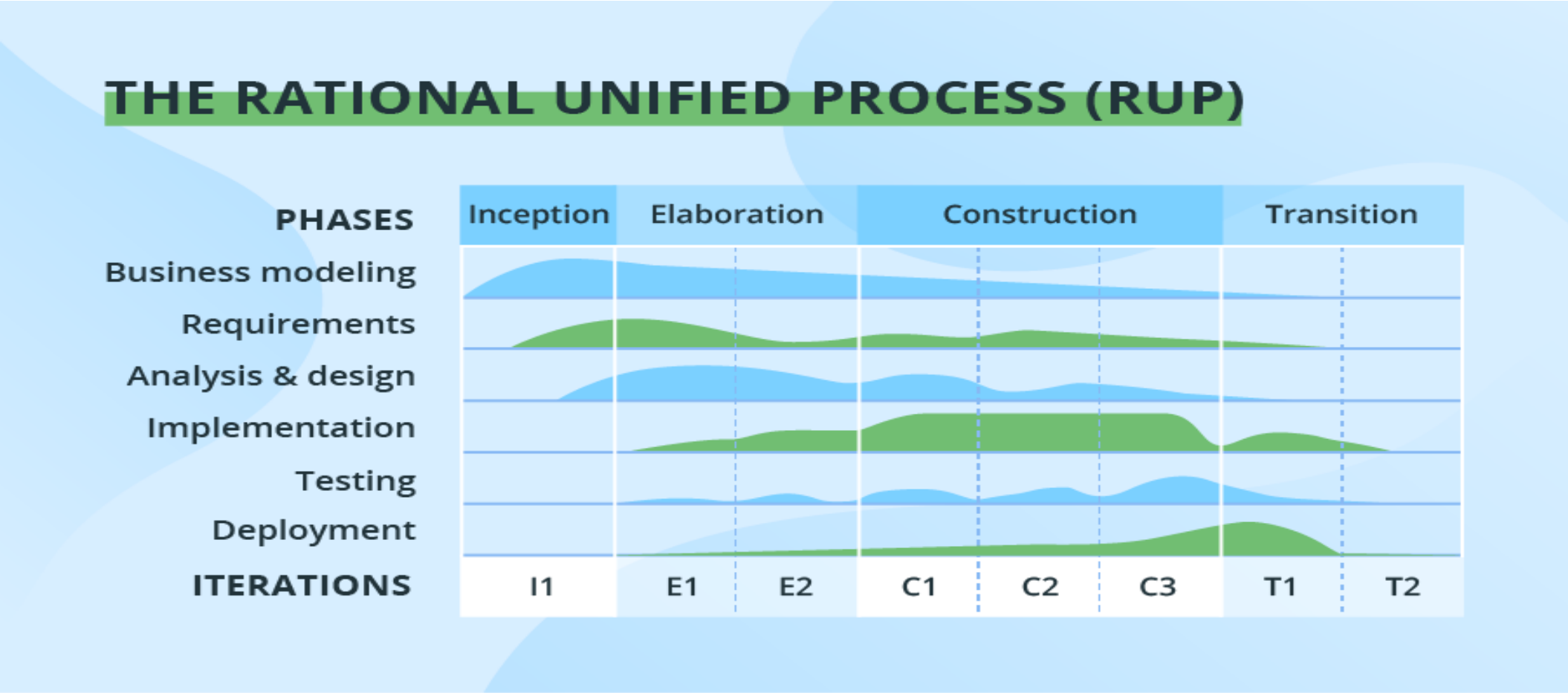
Object Oriented System Development Methodology

```
class HumanBeingClass
{
private int age;
private float height;
// public declarations of operations on HumanBeingClass
}// class HumanBeingClass
class ParentClass extends HumanBeingClass
{
private String nameOfOldestChild;
private int numberOfChildren;
// public declarations of operations on ParentClass
}// class ParentClass
```

Object Oriented System Development Methodology

- In the terminology of the object-oriented paradigm, there are **two** other ways of looking at the **relationship** between **Parent Class** and **Human Being Class.**
- We can say that **Parent Class** is a **Specialization** of **Human Being Class** or that **Human Being Class** is a **Generalization** of **Parent Class.**
- In addition, **classes have two other basic relationships: aggregation** and **association**.
- **Aggregation** refers to the ***components of a class***. For example, class **Personal Computer Class** might consist of components **CPU Class**, **Monitor Class**, **Keyboard Class**, and **Printer Class.**
- **Association** refers to a ***relationship of some kind between two apparently unrelated classes***. For example, there seems to be no connection between a **radiologist** and a **lawyer**, but a **radiologist** may **CONSULT** a **lawyer** for ***advice*** regarding a contract for **leasing a new MRI machine.**

Object Oriented System Development Methodology: *The Rational Unified Process (RUP)*



Object Oriented System Development Methodology: *The Rational Unified Process (RUP)*

- **RUP** is also a ***combination of linear and iterative frameworks***. It consists of ***several cycles***, each of which ***ends with the delivery of a product to the customer.***
- **The model divides** the software development process into **4 phases – inception, elaboration, construction, and transition**. **Each phase but Inception *(focusing on defining the project scope, business case, and initial requirements)* *is usually done in several iterations***.
- **All basic activities** (requirements, design, etc.) of the development process **are done in parallel across these 4 RUP phases**, though with different intensity.

Object Oriented System Development Methodology: *The Rational Unified Process (RUP)*

- During **INCEPTION**, ***a need or an idea is defined*** and ***its feasibility is evaluated.*** Evaluating what resources are needed, Risk assessments and project plans, vision or mission statements, and financial projections and business models
- The **ELABORATION PHASE** ***further evaluate the resources*** and ***costs needed for the project's full development***, ***creating actionable*** and ***executable baseline architecture of the software.*** Model is checked against milestone criteria and if it couldn't pass then again project can be canceled or redesigned. Use case model, Viable software architecture, Risk reduction plans, Use manual are done here.

Object Oriented System Development Methodology: *The Rational Unified Process (RUP)*

- The **CONSTRUCTION PHASE** *corresponds to the development processes* and *often takes the longest.* You create, write, collaborate and test your software focusing on the features and components of the system and how well they function. You start by incrementally expanding upon the baseline architecture, *building code* and *software until it's complete*. *Review the software user stability and transition plan before ending the RUP construction phase.*
- The **TRANSITION PHASE** *corresponds to the installation* and *post-development processes.* The final project is released to the public. Update project documentation. Beta testing is conducted. Defects are removed based on feedback from the public. Education and training.

Object Oriented System Development Methodology: *The Rational Unified Process (RUP)*

- The **requirements** ***are captured in the*** **use case model**. The **analysis model describes** ***the system as a set of*** **classes**. The **design model** ***defines the structure of the system as a set of*** **subsystems** and **interfaces.** The **deployment model** ***defines the distribution across***. The **implementation model** ***maps the classes to*** **components**. The **test model** ***verifies that the executable system provides the functionality described in the*** **use case model**.
- ***All models are related to each other through traceability dependencies. A model element can be traced to at least one element in an associated model****. For example,* **every use case** *has a* ***traceable relationship with at least*** **one class** *in the* ***analysis model****.* ***Traceability*** *allows to* ***understand the effect of change in one model on other models****, in particular it allows to provide traceability in the requirements.*

Object Oriented System Development Methodology: *The Rational Unified Process (RUP) Example Vs SDLC*

- **IBM Rational ClearCase**
  - **Purpose:** A *version control system designed to manage software development artifacts (like source code, documents, and test scripts) for large-scale software projects.*
  - **Why RUP Was Used:** IBM created RUP, so it was natural to apply it to complex software like ClearCase, which required robust architecture, risk management, and adaptability to evolving requirements.

Object Oriented System Development Methodology: *The Rational Unified Process (RUP) Example Vs SDLC*

- IBM Rational ClearCase

1. Inception Phase (Requirement Elicitation & Analysis)
   - Objective: *Define the project's scope, goals, and business case.*
   - Activities:
     - *Stakeholder Interviews & Workshops:* Gathered detailed requirements from software developers, project managers, and IT admins who would use ClearCase.
     - *Use Case Modeling:* Identified primary use cases, like *version control, branching, merging, and access control.*
     - *Risk Assessment:* Analyzed technical risks, such as *system scalability and integration challenges.*
   - Deliverables:
     - *Vision Document*
     - *Initial Use Case Model*
     - *Business Case*
     - *Risk List*

Object Oriented System Development Methodology: *The Rational Unified Process (RUP) Example Vs SDLC*

- IBM Rational ClearCase

2. **Elaboration Phase (Detailed Analysis & System Design)**
   - **Activities**:
     - ***Requirement Refinement:*** Expanded ***use cases to cover complex versioning scenarios.***
     - ***Architectural Design:*** Created an ***architecture blueprint for distributed version control, repository management, and access control.***
     - ***Prototyping:*** Built ***architectural prototypes to validate technical decisions.***
   - **Deliverables:**
     - ***Software Architecture Document (SAD)***
     - ***Updated Use Case Model***
     - ***Design Models (UML diagrams)***
     - ***Prototype Systems***

Object Oriented System Development Methodology: *The Rational Unified Process (RUP) Example Vs SDLC*

- **IBM Rational ClearCase**

3. **Construction Phase (Implementation & Development)**
   - **Objective:** Build the software product based on the refined design.
   - **Activities:**
   - ***Component Development:*** Developed core modules like ***repository services, version control APIs, and user interfaces.***
   - ***Iterative Development:*** Divided into mini-iterations, allowing ***continuous integration and feedback.***
   - ***Code Reviews & Pair Programming:*** Ensured code quality through collaborative development.

Object Oriented System Development Methodology: *The Rational Unified Process (RUP) Example Vs SDLC*

- IBM Rational ClearCase

3. Construction Phase (Implementation & Development)
   - Technologies Used:
     - *C/C++ for core system development*
     - *Java for integration tools*
     - *Shell scripting for automation*
   - Deliverables:
     - *Working Software Builds*
     - *Code Repositories*
     - *Technical Documentation*

Object Oriented System Development Methodology: *The Rational Unified Process (RUP) Example Vs SDLC*

- IBM Rational ClearCase

4. **Transition Phase (Testing & Deployment)**
   - **Objective:** Test the software thoroughly and deploy it for end-users.
   - **Activities:**
     - ***System Testing:*** Performed functional, regression, and performance testing.
     - ***User Acceptance Testing (UAT):*** Involved real users to validate features like version tracking and access controls.
     - ***Bug Fixes & Optimization:*** Addressed issues found during testing.
     - ***Deployment Planning:*** Prepared installation guides and deployment strategies.
   - **Deliverables:**
     - ***Final Software Product (Rational ClearCase)***
     - ***Test Reports***
     - ***Deployment Documentation***
     - ***User Manuals***