2.1 Software Process

The process that deals with the technical and management issues of software development is called software process. Clearly, many different types of activities need to be performed to develop software. As we have seen earlier, a software development project must have at least development activities and project management activities. All these activities together comprise the software process.

In this section we will define the major component processes of a software process and what their objectives are. Before, we do that, let us first clearly understand the three important entities that repeatedly occur in software engineering—software processes, software projects, and software products—and their relationship.

Processes, Projects, and Products

A software process, as mentioned earlier, specifies a method of developing software. A software project, on the other hand, is a development project in which a software is used. And software products are the outcomes of a software project. Each software development project starts with some needs and (hopefully) ends with some software that satisfies those needs.

One can view the software process as an abstract type, and each project is done using that process as an instant of this type. In other words, there can be many projects for a process (i.e., many projects can be done using a process), and there can be many products produced in a project. The relationship is shown in figure 2.1.

In this part, we will discuss the concept of the software processes, the components processes of a software process, and some models that have been proposed.
A software project is clearly a dynamic entity in which activities are performed, and project management process is needed to properly control this dynamic activity, so that the activities do not take the project astray but all activities are geared toward reaching the project goal.

The development process specifies the development and quality assurance activities that need to be performed, whereas the management process specifies how to plan and control these activities so that project objectives are met.

As the development process generally cannot handle change requests at an arbitrary point in time, to handle the inevitable change and rework requests another processes called software configuration control process, is generally used.

Overall for a particular project, to satisfy the objectives and handle the realities of software, at least three major constituent processes are needed: development process, project management process, and configuration control process.

These three constituent processes focus on the projects and the product. In fact, they can be considered as comprising product engineering processes, as their main objective is to produce the desired product. If the software process can be viewed as a static entity, then these three components processes will suffice. However, a software process itself is a dynamic entity, as it must change to adapt our increased understanding about software development and availability of newer technologies and tools. Due to this, a process to manage the software process is needed.

The process management process deals with the software process, its basic objective is to improve the software process. By improvement, we mean that the capability of the process to produce quality goods at low cost is improved.

### Software Process

A process usually involves a set of tools and techniques, as we defined them in Part I. Any process has the following characteristics:

- the activities are organized in a sequence, so that it is clear when one activity is performed relative to the other activities.
- every process has a guiding principles that explain the goals of each activity.
- constraints and controls may apply to an activity, resource, or product.

#### Software Life Cycle

In the early days of computing, software was developed by many individuals following their own methods. Often, the methods employed some form of "code and fix", where the programmer writes some code and then tests it to see how it performs. The programmer then uses the test results to modify or fix the code and tests again. Programmers were able to get by with this type of development for two reasons. First, no better way had been developed, and second, software was not that complex.

As software grew more complicated and organizations relied on computers for more of their operations, including finances and even human lives, this laissez faire approach to programming gave way to more disciplined methods. The overall framework in which software is conceived, developed, and maintained is known as the software development life cycle (SDLC). This section discusses the various types of SDLCs, along with its advantages and disadvantages.

*When a process involves the building of some product, we sometimes refer to the process as a life cycle. Thus, the software development process is sometimes called the software life cycle because it describes the life of a software product from its conception to its implementation, delivery, use, and maintenance.*

For the reminder of this Part, we examine a variety of software development process models, to see how organizing process activities can make development more effective.

#### 2.2 Software Process Models

Many process models are described in the software engineering literature. Some are prescriptions for the way software development should progress, and others are descriptions of the software development is done in actuality.

There are several reasons for modelling a process:

- when a group writes down a description of its development process, it forms a common understanding of the activities, resources, and constraints involved in software development.
- creating a process model helps the development team find inconsistencies, redundancies, and omissions in the process and in its constituent parts.
- the model should reflect the goals of the development, such as building high-quality software, finding faults early in development, and meeting required budget and schedule constraints.
- every process should be tailored for the special situation in which it will be used.

Every software development process model includes system requirements as input and a delivered product as output.
Software Life Cycle Models

A **life cycle model** defines the phases, milestones, deliverables, and evaluation criteria of the software development process. These form the basis of the **work breakdown structure (WBS)**, used for project planning and management.

Life cycles are usually referred to as models, and define the phases of a software development effort. Simple life cycles may have only three phases, Design, Development, and Maintenance; while complex life cycles may include 20 or more phases. Generally, software life cycles include the phases shown in Figure 2-3.

![Figure 2-3 Common Life Cycle Phases](image)

These "classic" phases are often divided into additional phases to allow better definition and control of the development process. They may also be repeated in an iterative manner, depending on the software complexity and the life cycle model used. Most life cycle phases are identical or similar to the common phases identified above and the following general descriptions will apply across most models. Note that single phases are composed of multiple activities.

**The Requirements Phase** consists of analyzing the problem or need for which the software is being developed. This analysis, a systems engineering activity, develops and specifies requirements, stating what the software must do. In addition to stated requirements, requirements are derived from higher-level requirements and statements of need.

In the **Design Phase** the software structure is defined. Technical approaches are chosen and problems are solved conceptually. This phase is often divided into a Preliminary Design Phase and a Detailed Design Phase. In the preliminary design the initial software architecture is developed. In the detailed design, functional modules are defined, along with user interfaces and interfaces between modules.

The **Implementation Phase** (sometimes called the Development Phase) is where the programming or coding takes place to execute the software design. This phase is often iterative, with unit and integration testing being performed after a software build, and the results used in another round of programming.

Software is tested for functionality and requirements compliance during the **Testing Phase**. Testing is often split into three separate phases: Unit Testing, Integration Testing, and Acceptance Testing.

During the **Deployment Phase** the software is installed in the intended system and users are trained in its operation. At this point the software development effort is considered complete.

The **Maintenance Phase** includes fixing errors and modifying or upgrading the software to provide additional functionality, such as enabling the software to work with new computing platforms. This phase costs far more in time and effort than the original development. Software maintainers must relearn the original software code to understand what was done, then make changes to specific modules to produce the desired effect without interfering with rest of the software. It's much easier to change requirements earlier than it is to change software code later. This also means that software should be developed with maintenance in mind.

### Waterfall Model

The waterfall model, also known as the linear sequential model, is shown in Figure 2-4 with its major phases, milestones, and products. It is a highly structured development process, first used on DoD software projects in the 1970s. It is the "traditional" approach to software development and was derived from defense and aerospace project lifecycles. It is considered superior to the previously used "code and fix" methods of software development, which lacked formal analysis and design.

![Figure 2-4 Waterfall Model](image)
An important consideration for the Waterfall model is that fixes or modifications are often put off until the maintenance phase. This can be very costly, as the cost to correct a problem gets higher with each successive phase.

**Advantages**
- System is well documented (i.e., easy to understand and implement)
- Phases correspond with project management phases.
- Cost and schedule estimates may be lower and more accurate.
- Details can be addressed with more engineering effort if software is large or complex.
- Widely used and known (in theory!)
- Works well on mature products and weak teams.

**Disadvantages**
- Difficult and expensive to make changes to documents, “swimming upstream”.
- Difficult to integrate risk management
- A working product is not available until late in the project.
- Progress and success are not observable until the later stages. If a mistake or deficiency exists in the documentation of earlier phases, it may not be discovered until the product is delivered.
- Significant administrative overhead, costly for small teams and projects.
- Doesn’t reflect iterative nature of exploratory development.

**Application**
The Waterfall model can be successfully used when requirements are well understood in the beginning and are not expected to change or evolve over the life of the project. Project risks should be relatively low.

**Incremental Model**
The incremental model is essentially a series of waterfall cycles. One variant is shown in Figure 2-5. The requirements are known at the beginning of the project and are divided into groups for incremental development. A core set of functions is identified in the first cycle and is built and deployed as the first release. The software development cycle is repeated, with each release adding more functionality until all requirements are met. Each development cycle acts as the maintenance phase for the previous software release. While new requirements that are discovered during the development of a given cycle can be implemented in subsequent cycles, this model assumes that most requirements are known up front. The effort is planned and executed to satisfy the initial list of requirements. A modification to the incremental model allows development cycles to overlap. That is, a subsequent cycle may begin before the previous cycle is complete.

![Figure 2-5 The Incremental Model is a Series of Waterfalls](image)

**Advantages**
- Provides some feedback, allowing later development cycles to learn from previous cycles.
- Requirements are relatively stable and may be better understood with each increment.
- Allows some requirements modification and may allow the addition of new requirements.
- It is more responsive to user needs than the waterfall model.
- A usable product is available with the first release, and each cycle results in greater functionality.
- The project can be stopped any time after the first cycle and leave a working product.
- Risk is spread out over multiple cycles.
- This method can usually be performed with fewer people than the waterfall model.
- Return on investment is visible earlier in the project.
- Project management may be easier for smaller, incremental projects.
- Testing may be easier on smaller portions of the system.

**Disadvantages**
- The majority of requirements must be known in the beginning.
- Formal reviews may be more difficult to implement on incremental releases than on a complete system.
- Because development is spread out over multiple iterations, interfaces between modules must be well-defined in the beginning.
- Cost and schedule overruns may result in an unfinished system.
- Operations are impacted as each new release is deployed.
- Users are required to learn how to use a new system with each deployment.

**Application**
The incremental model is good for projects where requirements are known at the beginning, but which need functionality early in the project or which can benefit from the feedback of earlier cycles. Because each cycle produces a working system, it may also be advantageous for projects whose continued funding is not assured and may be cut at any time. It is best used on low to medium-risk programs. If the risks are too high to build a successful system using a single waterfall cycle, spreading the development out over multiple cycles may lower the risks to a more manageable level.

**Evolutionary Model (Prototyping)**
The evolutionary model, like the incremental model, develops a product in multiple cycles. Unlike the incremental model, which simply adds more functionality with each cycle, this model produces a more refined prototype system with each iteration. The process, shown in Figure 2-6, begins in the center with initial requirements and plans, and progresses through multiple cycles of planning, risk analysis, engineering, and customer evaluation. Each cycle produces a prototype that the customer evaluates, followed by a refinement of requirements.
Specification, development, and testing activities are carried out concurrently (in the engineering quadrant) with rapid feedback. Since requirements continue to change, documentation is minimal, although essential information must still be included for understanding the system and for future support. Implementation compromises are often made in order to get the prototype working—permanent fixes can be made with the next prototype. Operational capability is achieved early, but users must be willing to learn how to use each new prototype.

General system requirements must be known prior to development. This is particularly helpful where evolving technology is being introduced into the project. The evolutionary model relies heavily on user feedback after each implementation to refine requirements for the next evolutionary step.

**Advantages**
- Project can begin without fully defining or understanding requirements.
- Final requirements are improved and more in line with real user needs.
- Risks are spread over multiple software builds and controlled better.
- Operational capability is achieved earlier in the program.
- Newer technology can be incorporated into the system as it becomes available during later prototypes.
- Documentation emphasizes the final product instead of the evolution of the product.
- This method combines a formal specification with an operational prototype.

**Disadvantages**
- Because there are more activities and changes, there is usually an increase in both cost and schedule over the waterfall method.
- Management activities are increased.
- Instead of a single switch over to a new system, there is an ongoing impact to current operations.
- Configuration management activities are increased.
- Greater coordination of resources is required.
- Users sometimes mistake a prototype for the final system.

**Application**
The evolutionary model can be employed on most types of acquisitions. However, it is usually employed on medium to high-risk systems. The evolutionary model should be considered for systems where requirements are not all known or not yet refined, but are expected to evolve. It is more applicable to new systems than upgrading existing software. The developing and using organizations must be flexible and willing to work with evolving prototypes.

Programs well suited to employ the evolutionary model have some or all of the following characteristics.
- Software intensive systems.
- Have a large number of diverse users.
- Have rapidly changing software technology.
- Developing an unprecedented system.
- Humans are an integral part of the system.
- Limited capability is needed quickly.

---

**Spiral Model**
The spiral model was developed with the goal of reducing risk in the software life cycle. It combines elements of the waterfall, evolutionary, and incremental models, and depending on how it is implemented can strongly resemble any combination of the others. The model's spiral nature can be seen in Figure 2-7, one of several variants. The project starts at the center and progresses through multiple cycles, each working through the software development activities associated with the four quadrants:
- Determine objectives, alternatives, constraints.
- Evaluate alternatives. Identify and resolve risks.
- Develop the next-level product.
- Plan the next phase.

Risk management is a key element of the Spiral model and each round of the spiral identifies problems with the highest risk and develops solutions for that set of problems. The process may even resemble a waterfall with additional risk management techniques. Each cycle ends in a review in which stakeholders agree on plans for the next cycle. While a prototype may be produced for IOC, software is usually not developed for release until the last cycle.

**Advantages**
- It provides better risk management than other models.
- Requirements are better defined.
- System is more responsive to user needs.

**Disadvantages**
- The spiral model is more complex and harder to manage.
- This method usually increases development costs and schedule.

**Application**
The spiral method should be considered for projects where risks are high, requirements must be refined, and user needs are very important.
Agile

Software Projects (1980’s – Early 1990’s). Long lived projects, large systems, large number of software components and developers. Best way to achieve successful development. Careful planning, formalized quality assurance, analysis, design, and controlled software development process. Also, requirements are less likely to change. (for example: Critical Systems)

Software Projects (Recent). As computer technology weaves into society requirements for many small and medium size software systems arises. In addition, the dynamic nature of software human requirements make software development more challenging (not necessarily technical difficulties but unstable requirements).

So, we need software process models that are more amenable to changes and more flexible.

In the late 1990s, some developers formulated their own principles, trying to highlight the roles that flexibility could play in producing software quickly and capably. They codified their thinking in an “agile manifesto” that focuses on four tenets of an alternative way of thinking about software development (Agile Alliance 2001)

- They value individuals and interactions over process and tools. This philosophy includes supplying developers with the resources they need and then trusting them to do their jobs well. Teams organize themselves and communicate through face-to-face interaction rather than through documentation.
- They prefer to invest time in producing working software rather than in producing comprehensive documentation. That is, the primary measure of success is the degree to which the software works properly.
- They focus on customer collaboration rather than contract negotiation, thereby involving the customer in key aspects of the development process.
- They concentrate on responding to change rather than on creating a plan and then following it, because they believe that it is impossible to anticipate all requirements at the beginning of development.

The overall goal of agile development is to satisfy the customer by “early and continuous delivery of valuable software” (Agile Alliance 2001).

What is agility? Agile software team is a nimble team able to appropriately respond to changes.

What kind of changes? Changes in team members, Changes in software being built, and Changes because of new technology.

Agile: A conceptual framework generally centered on iterative and incremental delivery of working software, driven by the customer. The iterative part suggests that we are repeating, or iterating, a complete lifecycle of development over a short, fixed span of time. With each of these iterations, we ship some working subset, or increment, of features.

Agile methodologies generally refer to software development methodologies that focus on bringing value to the customer as quickly as possible and the ability to respond to change over up-front documentation and long-term planning. As defined by the Agile Manifesto agile development means.

Manifesto for Agile Software Development
We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.

Agile Methods

Several agile methods exist. Some of the more well known include extreme programming (XP), lean software development, Crystal, DSDM (Dynamic Systems Development Method), Scrum, and feature-driven development (FDD).

- Lean—Move closer to customer, shorter cycles, eliminate waste, decide as late as possible, empower the team, build in integrity
- SDM—Empower the team to make decisions, emphasize frequent product delivery, integrate testing throughout, promote collaboration and cooperation between all stakeholders
- FDD—Center development around the feature, create a domain model with domain experts
- Crystal—Emphasize people, gather techniques from other methods, improve communications, adapt the process itself (shrink or grow to fit)
- Scrum—Manage a prioritized list of requires on a product backlog, collaborate through daily standup meetings, exhibit the product upon iteration completion, use retrospectives to correct the process
- XP—Emphasize the values of communication, simplicity, feedback, and courage; use specific technical and collaborative practices, including Test-Driven Development (TDD), refactoring, pair programming, continuous integration, open workspace, and automated acceptance tests

Agile Process Models – Principles (Rules)

Agile project methodologies help to communicate effectively and to ensure the project integrates with other parties’ deliverables. Its main principles are:

<table>
<thead>
<tr>
<th>Principle</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer involvement</td>
<td>The customer should be closely involved throughout the development process. Their role is to provide and prioritize system requirements and to evaluate the iterations of the system.</td>
</tr>
<tr>
<td>Incremental delivery</td>
<td>The software is developed in increments with the customer specifying the requirements to be included in each increment.</td>
</tr>
</tbody>
</table>
People not process | The skills of the development team should be recognized and exploited. The team should be left to develop their own ways of working without prescriptive processes.
---|---
Embrace change | Expect the system requirements to change and design the system so that it can accommodate these changes.
Maintain simplicity | Focus on simplicity in both the software being developed and in the development process used. Wherever possible, actively work to eliminate complexity from the system.

Some agile process models (Extreme programming)
The most widely used agile process, originally proposed by Kent Beck. Perhaps the best-known and most widely used agile method.

- Extreme Programming (XP) takes an ‘extreme’ approach to iterative development.
- New versions may be built several times per day;
- Increments are delivered to customers every 2 weeks;
- All tests must be run for every build and the build is only accepted if tests run successfully.

2.3 Tools and Techniques for Process Modelling
There are many choices for modelling tools and techniques, once you decide what you want to capture in your process model; we have seen several modelling approaches in our model depictions in the preceding section. The appropriate technique for you depends on your goals and your preferred work style.

Desirable Properties of Process Modelling Tools and Techniques
There are many process modelling tools and techniques, and researchers continue to work to determine which ones are most appropriate for a given situation. But there are some characteristics that are helpful, regardless of techniques. Curtis, Kenilner, and Över (1992) have identified five categories of desirable properties:

- **Facilities human understanding and communication.** The technique should be able to represent the process in a form that most customers and developers can understand, encouraging communication about the process agreement on its form and improvements.
- **Supports process improvement.** The technique should identify the essential components of a development or maintenance process.
- **Supports process management.** The technique should allow the process to be project-specific.
- **Provides automated guidance in performing the process.** The technique should define all or part of the software development environment, provide guidance and suggestions, and retain reusable process representations for later use.
- **Supports automated process execution.** The technique should automate all or part of the process, support cooperative work, capture relevant measurement data, and enforce rules to ensure process integrity.

These characteristics can act as useful guidelines for selecting a process modeling technique for your development project.