Software Engineering Processes

In order for software to be consistently well engineered, its development must be conducted in an orderly process. It is sometimes possible for a small software product to be developed without a well-defined process. However, for a software project of any substantial size, involving more than a few people, a good process is essential. The process can be viewed as a road map by which the project participants understand where they are going and how they are going to get there.

There is general agreement among software engineers on the major steps of a software process. Figure 1 is a graphical depiction of these steps. The first three steps in the process diagram coincide with the *basic steps of the problem solving process*, as in Table 4.

The fourth step in the process is the ***post-development phase***, where the product is deployed to its users, maintained as necessary, and enhanced to meet evolving requirements.

The first two steps of the process are often referred to, respectively, as the "***what and how***" of software development. The "**Analyz e and Specify**" step defines *what* the problem is to be solved; the "**Design and Implement**" step entails *how* the problem is solved.

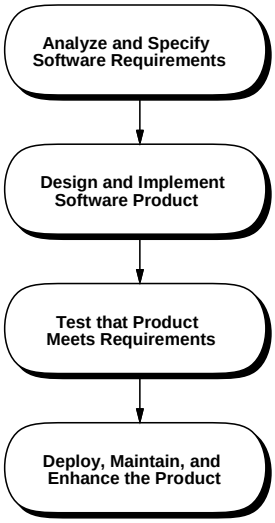


Figure 1: Diagram of general software process steps.

Problem-Solving Phase	Software Process Step
Define the Problem	- Analyze and Specify Software Requirements
Solve the Problem	-Design and Implement Software Product
Verify the Solution	-Test that Product Meets Requirements

Table 4: Correspondence between problem-solving and software processes.

-While these steps are common in most definitions of software process, there are wide variations in how process details are defined.

-The variations stem from the ***kind of software being developed*** and the ***people doing the development***.

Example,

> *The process for developing a well-understood business application with a highly experienced team can be quite different from the process of developing an experimental artificial intelligence program with a group of academic researchers.*

-There is variation in terminology, how processes are structured, and the emphasis placed on different aspects of the process.

-Independent of technical details, there are ***general quality criteria that apply to any good process***. These criteria include the following:

1. The process is suited to the people involved in a project and the type of software being developed.
2. All project participants clearly understand the process, or at minimum the part of the process in which they are directly involved.
3. If possible, the process is defined based on the experience of engineers who have participated in successful projects in the past, in an application domain similar to the project at hand.
4. The process is subject to regular evaluation, so that adjustments can be made as necessary during a project, and so the process can be improved for future projects.

-Developing software often involves people of diverse backgrounds, varying skills, and differing viewpoints on the product to be developed. Added to this are the facts that software projects can take a long time to complete and cost a lot of money. Given these facts, software development can be quite challenging, and at times trying for those doing it.

-Having a well-defined software process can help participants meet the challenges and minimize the trying times. However, any software process must be conducted by people who are willing and able to work effectively with one another. Effective human communication is absolutely essential to any software development project, whatever specific technical process is employed.

General Concepts of Software Processes

Process Terminology

The following terminologies are used in regard to software processes:

- ***software process:*** a hierarchical collection of *process steps*; hierarchical means that a process step can in turn have *sub-steps*
- ***process step:*** one of the activities of a software process, for example "**Analyz e and Specify Software Requirements**" is the first step in Figure 1 ; for clarity and consistency of definition, process steps are named with verbs or verb phrases

- ***software artifact:*** a software work product produced by a process step; for example, a requirements specification document is an artifact produced by the "**Analyz e and Specify**" step; for clarity and consistency, process artifacts are named with nouns or noun phrases
- ***ordered step:*** a process step that is performed in a particular order in relation to other steps; the steps shown in Figure 1 are ordered, as indicated by the arrows in the diagram
- ***pervasive step:*** a process step that is performed continuously or at regularly-scheduled intervals throughout the ordered process; for example, process steps to perform project management tasks are pervasive, since management is a continuous ongoing activity
- ***process enactment:*** the activity of performing a process; most process steps are enacted by people, but some can be automated and enacted by a software development tool
- ***step precondition:*** a condition that must be true before a process step is enacted; for example, a precondition for the "**Design and Implement**" step could be that the requirements specification is signed off by the customer
- ***step postcondition:*** a condition that is true after a process step is enacted; for example, a postcondition for the "**Design and Implement**" step is that the implementation is complete and ready to be tested for final delivery.

Process Structure

This refers to mechanism of depicting software process.

There are a variety of ways to depict a process. A typical ***graphical depiction*** uses a diagram with boxes and arrows, as shown in Figure 1. In this style of diagram, *a process step is shown as a rounded box,* and the *order of steps is depicted with the arrowed lines*.

-Process sub-steps are typically shown with a box expansion notation. For example, Figure 2 shows the expansion of the "**Analyz e and Specify**" step. The activities of the first sub-step include general aspects of requirements analysis, such as defining the overall problem, identifying personnel, and studying related products.

-The second sub-step defines functional requirements for the way the software actually works, i.e., what functions it performs. (*Functional requirements define the specific functions that the software performs, along with the data operated on by the functions*). The last sub-step defines non-functional requirements, such as how much the product will cost to develop and how long it will take to develop. (*These requirements address aspects of the system other than the specific functions it performs.*)

-Aspects include system performance, costs, and such general system characteristics as reliability, security, and portability. The non-functional requirements also address aspects of the system development process and operational personnel.

There are three major categories of non-functional requirements, corresponding to the three sub-steps of the non-functional process:

- ***system-related*** -- these are non-functional requirements related to the software itself, such as performance, operational environment requirements, product standards, and general system characteristics
- ***process-related*** -- these are requirements for the software development process, including how long

it will take, how much it will cost, and other relevant matters

- ***personnel-related*** -- these are requirements related to the people involved with software development and its use

-A more compact process notation uses mostly text (***text depiction***), with indentation and small icons to depict sub-step expansion. Figure 3 shows a textual version of the general process, with the first step partially expanded, and other steps unexpanded. Right-pointing arrowheads depict an unexpanded process step. Down-

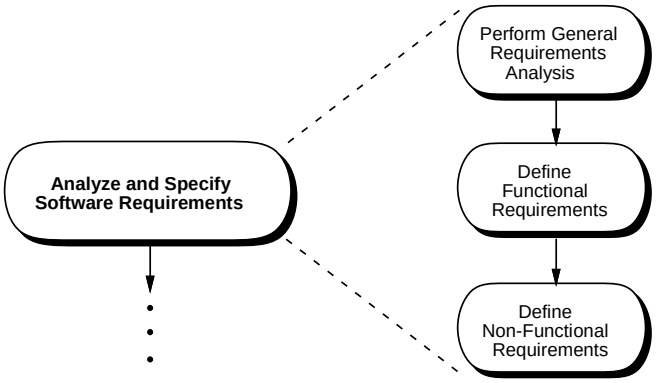


Figure 2: Expansion of the ‘‘Analyze and Specify’’ Step.

Analyze and Specify SoftwareRequirements PerformGeneral Requirements Analysis

State Problem to be Solved
Identify People Involved
Analyze Operational Setting
Analyze Impacts
Identify Positive Impacts
Identify Negative Impacts
Analyze Related Systems
Analyze Feasibility

Define Functional Requirements
Define Non-Functional Requirements
Design and Implement Software Product
Test that Product Meets Requirements
Deploy, Maintain, and Enhance the Product

Figure 3: Textual process depiction.

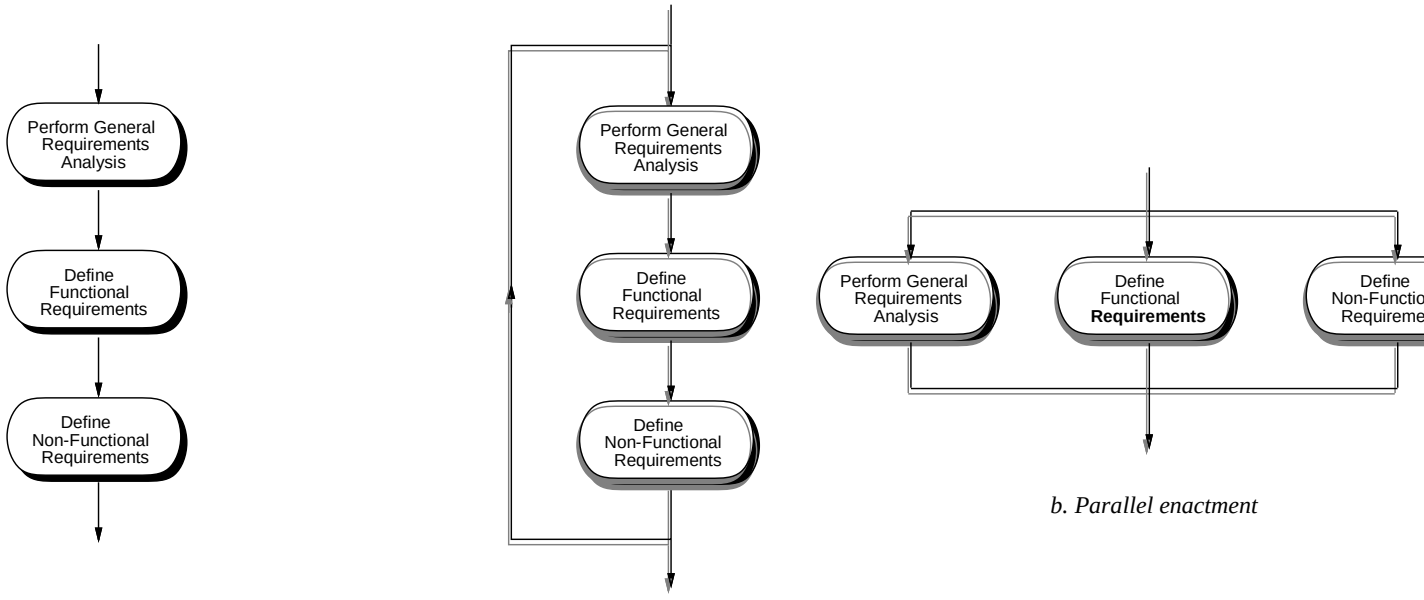
pointing arrowheads depict a process step with its sub-steps expanded immediately below. A round bullet depicts a process step that has no sub-steps.

-Depending on the context, one or the other form of process depiction can be useful. When the emphasis is on the flow of the process, the graphical depiction can be most useful. To show complete process details, the textual depiction is generally more appropriate.

Styles of Process Enactment

-Once the steps of a software process are defined, they can be enacted in different ways. The three general forms of ordered enactment are *sequential*, *iterative*, and *parallel*. These are illustrated in Figure 4 for the three sub-steps of the**Analyz e and Specify** step.

-**Sequential enactment** means that the steps are performed one after the other in a strictly sequential order. A preceding step must be completed before the following step begins. For the three steps in Figure a, this means that the general analysis is completed first, followed by functional requirements, followed by nonfunctional requirements.



a. Sequential enactment

b. Iterative enactment

b. Parallel enactment

Figure 4: Three styles of enactment.

Iterative enactment follows an underlying sequential order, but allows a step to be only partially completed before the following step begins. Then at the end of a sequence, the steps can be re-enacted to complete some additional work. When each step is fully completed, the entire sequence is done.

-In Figure b, some initial work on general analysis can be completed, enough to start the function requirements analysis. After some functional requirements are done, work on the non-functional requirements can begin. Then the three steps are repeated until each is complete.

Parallel enactment allows two ore more steps to be performed at the same time, independent of one another. When the work of each step is completed, the process moves on to the subsequent steps.

NB

Which of these enactment styles to use is determined by the mutual dependencies among the steps. For some projects, the determination may be made that a complete understanding of the general requirements is necessary before the functional and non-functional requirements begin. In this case, a strictly sequential order is followed. In other projects, it may be determined that general requirements need only be partially understood initially, in which case an in iterative order is appropriate.

Hybrid style of enactment

-Occurs when a combination of any of the three major forms of enactment is followed. e.g

> *-a first pass at general analysis is performed. Then the functional and non-functional analyses proceed in parallel. The process then iterates back to refine the general requirements and then proceed with further functional and non-functional refinements*.

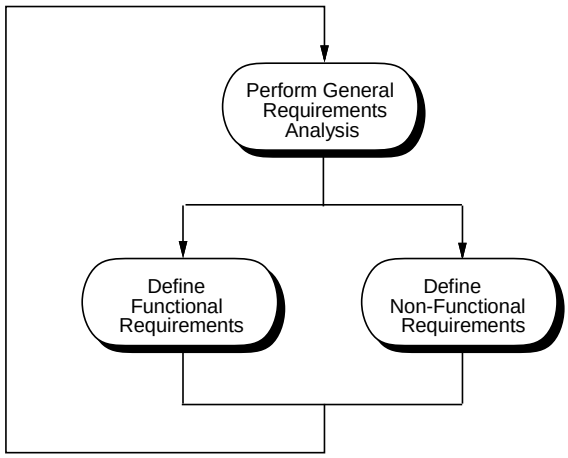


Figure 5: Hybrid process enactment.

Pervasive enactment

A fourth kind of enactment is *pervasive*. The three styles of process enactment discussed so far apply to process steps that are performed in some order relative to one another.

-***A pervasive process step is performed continuously throughout the entire process or at regularly scheduled points in time.*** A good example of pervasive process steps are those related to project management. A well managed project will have regularly-scheduled meetings that occur on specific scheduled dates, independent of what specific ordered step developers may be conducting. The steps of the process dealing with project supervision occur essentially continuously, as the supervisors oversee developer's work, track progress, and ensure that the process is on schedule.

Testing is another part of the software process that can be considered to be pervasive. In some traditional models of software process, testing is an ordered step that comes near the end, after the implementation is complete.

Ordered Process Steps

-The ordered steps can be viewed as a process of successive refinement, from an initial idea through to a deployable software product. In this sense, the process is based on a ***divide and conquer strategy***, where the focus of each step is a particular aspect of the overall development effort.

-At each step, the developers have the responsibility to focus on what is important at that level of refinement. They also have the freedom to ignore or give only limited consideration to what is important at other levels of refinement.

-The focus of the **Analyz e** step is on user requirements. The needs of the user are the primary concern at this level. Concern with details of program design and implementation should be limited to what is feasible to implement.

-The focus of the **Specify** step is on building a "real-world" software model. Real-world in this context means that the model defines the parts of the software that are directly relevant to the end user, without program implementation details.

-The focus of the **Prototype** step is building a partially operational program as rapidly as possible. The purpose of the prototype is to help solidify everyone's understanding of the requirements. For the users, the prototype provides a concrete view that can help them focus on what the software will do. For the developers, the prototype helps them explore concrete ideas for functionality and human-computer interface. In building the prototype, the developers need to be free to employ whatever ***techniques support very rapid results.*** This generally means ignoring important but time-consuming steps of design and implementation that must be followed when building the full-scale, production-quality product.

-The focus of the **Design** step is the overall architecture of the program, based on the results of the previous steps of the process. A high-level architecture defines large-grain program units and the

interconnection between the units (*The high level architecture of a program is defined in terms of data classes and computational functions.*). A lower-level architecture defines further details, down to the level of the application program interface (API). The designers do not to focus on lower-level implementation details, such as concrete data structuring and the procedural implementation of program functions.

-The focus of the **Implement** step is the algorithmic and data detail of an efficient program. The implementors assume that previous steps have been conducted properly, so there will be a small number or problems that need to be addressed at the previous levels while the implementation is under way. In terms of the original idea of a problem-solving process, the implementors are free to assume that the problem has been well defined before they implement its solution.

-The focus of the **Deploy** step is to put the developed product to use. This entails distribution, installation and, as necessary, maintenance. The maintenance may be carried out by a separate post-develop team, by the original developers, or by some combination of these. Users and maintainers alike should have the freedom to assume that the developers have built a quality product, that will work correctly and meet the users' needs.

Step	Responsibilities	Freedoms
Analyz e	Understand the human users and their needs; define the human-computer-interface (HCI).	Ignore program design and implementation details as much as possible.
Specify	Define a real-world software model; specify the behavior of all user-level operations and user-visible objects.	Ignore concrete implementation details; ignore programming language details.
Prototype	Rapidly develop the prototype.	Ignore time-consuming software design and implementation methods; ignore program efficiency.
Design	Define the architectural organization of the program; define the application programmer interface (API); work with the analysis team to address problems in the requirements or specification.	Assume that user-level requirements have been properly analyzed and specified, such that there will be a small number requirements problems; ignore lowlevel details of program implementation.
Implement	Implement the design as an efficient program; work with the analysis team on problems in the requirements or specification; work with the design team on problems in the design.	Assume that the previous steps have been carried out properly, such that there will be a small number of higher-level problems that need to be addressed.
Deploy	Install and configure the program for use; report problems to the maintenance staff.	As an end user, ignore internal details of the program and how it was developed; as both maintainer and user, assume that the developers have built a quality product, such that there will be few problems that need to be addressed.

Summary :Responsibilities and freedoms of the ordered process steps.

Pervasive Process Steps

-A pervasive step is performed continuously throughout the ordered process, or at regularly scheduled points in time.

In some cases, the work performed in a pervasive step applies generically to all of the ordered steps. For example, many of the tasks performed to manage a software project apply uniformly across all of the ordered steps. In other cases, the work performed in a pervasive step must be tailored individually to the different ordered steps. For example, the testing of an implemented program has specific details that are different than the testing of the requirements.

The particular aspects of pervasive processing that are uniform across ordered steps are the following:

- most aspects of the **Manage** step are generic for all ordered steps;
- testing based on human inspection is generic for all ordered-step artifacts, however other aspects of testing are artifact-specific;
- basic documentation practices are generic for all artifacts, but specific content obviously varies.

Testing

-The major types of testing that are applicable to some or all software artifacts includes:

Inspection testing- This is carried out by humans, who thoroughly and systematically inspect software artifacts. To do so, the inspectors define a ***testing plan*** that specifies what inspection test standards need to be applied to the artifacts being tested.

-For example, inspection testing the requirements entails activities such as careful proof reading of the prose, ensuring the figures are clear, and enforcing any domain-specific constraints for the requirements. Inspecting an implementation involves careful and thorough review of program code, typically performed by someone other than the code's author.

-Inspection tests can be performed by individuals, as well as by groups. Group testing can be conducted in informal walkthroughs and in formal reviews. The members of review groups are all stakeholders for whom the inspection is relevant. The results of inspection tests are recorded and the appropriate authorities sign off that the tests have been conducted successfully. For example, a customer with authority to sign off on requirements does so at the culmination of a successful requirements inspection review. Any inspection test failures are recorded in the test record, for subsequent repair.

Analytic validation -This is typically performed using automated analysis tools, frequently in conjunction with an artifact build.

Static analysis is performed to validate the structure of an artifact.

> Probably the most typical form of static analysis is that performed by a compiler on program source code. Static analysis tools are also available for other artifacts, such as spelling and grammar checking for requirements documents. Syntactic and semantic analysis tools are available for formal software models and designs.

Dynamic analysis validates the dynamic behavior of an executable artifact, most typically the implementation. For example, dynamic analysis tools can be used to validate the behavior of distributed and parallel programs, checking for such problems as deadlock or race conditions. Dynamic analysis is also used to validate that system performance is adequate to meet stated performance requirements.

Functional testing This is performed on the operational program produced by the **Implement** step. Functional testing is carried out at three levels of program structure:

- unit testing of program functions (i.e., methods)
- module testing of program classes (or other encapsulation constructs)
- integration testing among the classes

At each of these levels, the testing steps entail defining the test plans, performing the tests, recording the testing results, and validating that the results are correct.

Acceptance testing This is conducted by end users, or their representatives, on the delivered software product. It entails the same sub-steps as functional testing, namely plan, perform, record, and validate. Whereas functional tests are defined in terms of the program implementation, acceptances tests are defined strictly at the end-user level of functionality. When validated, acceptance tests signal that the product is ready to be released to the full user community.

Well-Known Process Models- (Waterfall, spiral, vmodel, agile and cleanroom models)

-Taken together, the steps of a process and its style of enactment can be considered a *process model*. Overall, there is a clear thread of commonality among various process models. Virtually all models include requirements gathering, design, implementation, and testing as parts of the process.

-What distinguishes the models is less about the steps themselves than the manner in which they are enacted and the relative emphasis placed on each step. What distinguishes the models also has little to do with their catchy names. For example, the most salient feature of the "spiral model" is not its shape, but rather its focus on continual risk assessment throughout the process.