

## Software engineering reference framework

***Citation for published version (APA):***

Chaudron, M. R. V., Groote, J. F., Hee, van, K. M., Hemerik, C., Somers, L. J. A. M., & Verhoeff, T. (2004). *Software engineering reference framework*. (Computer science reports; Vol. 0439). Technische Universiteit Eindhoven.

***Document status and date:***

Published: 01/01/2004

***Document Version:***

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

***Please check the document version of this publication:***

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

***General rights***

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

[www.tue.nl/taverne](http://www.tue.nl/taverne)

***Take down policy***

If you believe that this document breaches copyright please contact us at:

[openaccess@tue.nl](mailto:openaccess@tue.nl)

providing details and we will investigate your claim.

# Software Engineering Reference Framework

**Michel Chaudron, Jan Friso Groote, Kees van Hee,  
Kees Hemerik, Lou Somers, Tom Verhoeff.**

Department of Mathematics and Computer Science  
Eindhoven University of Technology  
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

{M.R.V.Chaudron, J.F.Groote, K.M.v.Hee, C.Hemerik, L.J.A.M.Somers, T.Verhoeff}@tue.nl

## 1. Objectives of the framework

This Software Engineering Reference Framework is meant for the education of computer science students at Eindhoven University of Technology. The framework will be used to unify the basic concepts and the terminology in the various courses that cover topics of software engineering and in the OGO-projects, including the Software Engineering Project.

The reference framework is not an attempt to present a “one size fits all” approach, that should be adopted by everyone. Therefore, it does not prescribe *how* systems should be developed or maintained. It is meant to relate topics of different courses in order to give students a better insight in the structure of computer science and software engineering. Parts of the terminology and the structure are derived from the IEEE software engineering standards [IEEE].

## 2. Scope of the framework

We start with some elementary concepts. Software engineering is concerned with the development and maintenance of software systems by the systematic application of engineering techniques in the software domain.

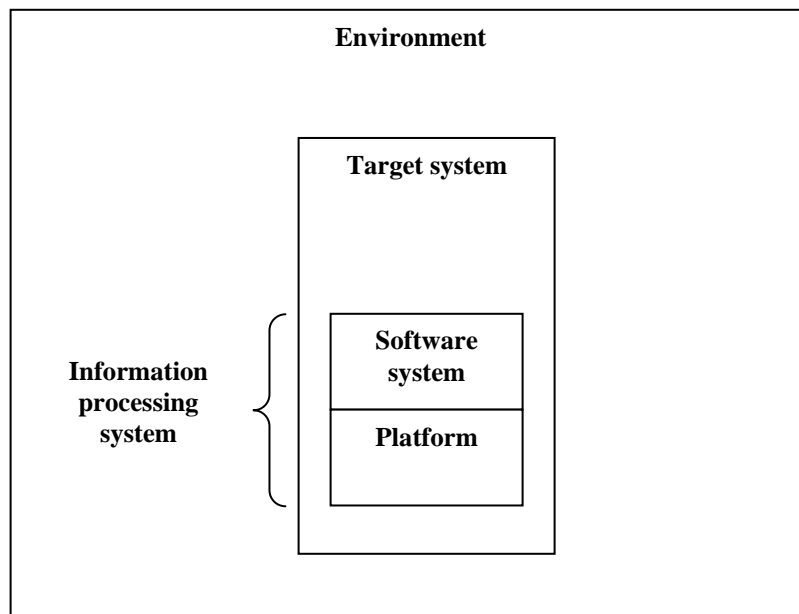
The process that creates the system is called the *development* process and the process that keeps the system in operation is called the *maintenance* process. We focus on the development process here, since the maintenance process is a sequence of (re)development processes. Together they cover the *life cycle* of a system. A process recursively consists of sub-processes and the atomic sub-processes are called *tasks*. A task has a clear goal with well-defined input and output; the latter is also called the *deliverable*. There are two kinds of deliverables: documents that describe the status and progress of the development *process* and documents (including program code) describing the *system* to be developed. We

call the first kind the (*process*) reports and the second kind the (*system*) artefacts. We focus on the artefacts that make up the system and not on the tasks or sub-processes that produce these artefacts.

The framework contains the following topics:

1. Definitions of the *artefacts* that make up a system (e.g. program code, requirements and manuals)
2. Basic concepts of *project management* (concepts like: scoping, planning, milestones and deliverables)
3. The most common *process models*, i.e. the ordering principles of the tasks (strategies like: waterfall, evolutionary and incremental)
4. The most common *methods* and *techniques* to fulfil these tasks (not in this report)
5. A glossary of *terminology*

Terms in italics are defined in the glossary. The following picture shows how a system fits in its context.



The *target system* is the for which software needs to be designed or redesigned. In case of an embedded system, a video recorder is an example of a target system and in case of an information system a business unit of an insurance company is an example. In the target system, an *information processing system* has to support the processes of the target system. The information processing system consists of

a *software system* we have to develop and a *platform*, consisting of *computer hardware* and *basic software* (like an operating system) that is available. It is in many cases preferable if the software engineer is involved in the (re)design of the target system as a whole, although the software system is the one to be developed.

### 3. Artefacts

The artefacts are grouped into coherent clusters. For example “inception” is a group of artefacts that describes the background of the system. It does not mean that the artefacts in a cluster have to be produced as one uninterrupted task. The artefacts are the important objects. For some clusters it is quite natural to produce them in one task, while others, in particular *verification*, is usually spread over several tasks. The order of artefacts is not arbitrary since in most cases one needs the previous artefacts in order to make them.

Depending on the typical system, it is possible to combine artefacts or to split them. Note that each artefact is documentation (a design, requirements, program code, installation procedure). In all cases, it is necessary to incorporate the motivation for choices made and to give sufficient explication.

#### Inception

Inception is the first step in designing or adapting a system. It answers the question: “Why do we need a new or renewed system?”

- Mission statement:  
The system has to be developed for a particular purpose. This artefact describes the purpose of the system from the perspective of the stakeholders.
- Context analysis:  
This artefact consists of a description of the target system and its environment for which the software system has to be developed, often in terms of the stakeholders. This description can include the actors and (models of) the processes and objects in the target system and its environment, that are relevant for the software system to be developed.
- Feasibility study:  
The analysis of technical and economical feasibility of the system. The last one leads to the so-called *business case* for the system.

In the artifacts above, the software system is to be treated as a black box. Its internal structure is not referred to.

#### Requirements

Requirements engineering answers the question: “What should the system do for

its environment?" We distinguish three types of requirements.

- User requirements:  
The description of the functions that the system has to fulfil for its environment in terms of the users interacting with the system, e.g. in the form of *use cases*.
- Software requirements:  
We distinguish *functional* and *extra-functional requirements* of the software system. The software requirements are a translation and a more precise description of the user requirements, in terms for software engineers.
- Development requirements:  
Requirements on the means with which the system should be realized. In particular the documentation to be delivered, the development methods and tools to be used and the implementation *platform*.

These three artefacts are often considered as the contract between the principal stakeholder and the software engineers.

## Design

The design answers the question: "What are the constituent concepts of the system?"

- Software architecture:  
The software architecture consists of:
  - The structure, i.e. a set of *components* and relationships between these components, such as *interaction* relations.
  - For each component, a model of its behaviour and its logical data structure, from an external viewpoint.
  - For each interaction relation, a model of the *interface*.

It is often necessary to describe several independent views in a software architecture, such as a logical view or a physical view.

- Detailed design:
  - The translation of the software architecture into the constructs of the implementation technology, such as modules, procedures and classes or objects.
  - The algorithms and data structures that realize the behavior and data structures of the components.

Note that the derivation of the program structure as well as the algorithms and data structures may be a process of stepwise refinement.

## Construction

The construction answers the question: "What is the system exactly?"

It consists of:

- Program code:  
For new components of the system there should be source code. For *OTS (Off-The-Shelf) components* that were already available this is not always possible. A reference to the source code and an executable are needed for these components. All code must be clearly annotated to clarify its functionality and to express the relations between pieces of code, and between the code and other system artefacts.
- Configuration parameters:  
Larger components and in particular *OTS components* may have configuration parameters. The values of these parameters are essential elements of the system. The relationship between the required functionality, the provided functionality and the values of the configuration parameters must be well documented.

## Verification

Verification answers the question: “Does the system conform to its specifications?” It determines consistency among the artefacts. The question is answered in steps, e.g. “is the architecture conform the requirements?”, “is the detailed design consistent with the architecture?”, “are the algorithms correct?” and “is the program code implementing the design correctly?”. The verification whether the complete system satisfies the user requirements, is called *validation*. We distinguish three approaches to verification:

- Reviews:  
Reviews are performed by experts. They give their expert judgement on any artefact. There are several ways to perform the inspections and often check lists are used as guidance.
- Proofs:  
Some properties can be verified formally, provided that the property can be formalized. Typical examples are the verification of invariance properties and the absence of deadlocks. Software tools such as model checkers and theorem provers, may be used to provide the proofs,.
- Testing:  
A test consists of a:
  - Test specification: What will be tested?
  - Test script: How will it be done?
  - Test cases: The test scenario’s with input data and expected results.
  - Test criteria: What are the criteria to determine that the test is successful?
  - Test report: A summary containing the outcomes of the different tests and the overall conclusion.

We distinguish several kinds of tests:

- Component tests: Testing a component against its specification.
- Integration tests: Testing the interaction between the components against the architecture.
- System tests: Testing the integrated system against the use case scenarios.

Verification generally relates two or more artefacts. Therefore, there will be many verification artefacts, that can exist as separate documents, but can also be collected in an overall verification artefact, or be included as appendices in the verified artefacts.

### **Deployment**

Deployment refers to the transition of the system from its development phase to the operational phase in the life cycle of a system. Typical ingredients are:

- Build and installation procedures.
- Conversion software to transform the data of the old system into the new one.
- Instructions and manuals for users and administrators.
- Plan for the maintenance process.

## **4. Project management**

For project management there are standardized methods such as PRINCE2, Projects In Controlled Environments, from the UK Office of Government Commerce [PRINCE2]. There is also a framework called, PMBOK, Project Management Body of Knowledge, from the USA based PMI, Project Management Institute [PMBOK]. Both have a lot in common. Since we use only the basics of project management we are compliant with each of them. We assume that projects always take place in a customer-supplier environment, i.e. there are two roles: the customer, who orders the development of a system and the supplier who will develop the system.

Project management has three important components:

1. Project organization, consisting of persons and teams having specific roles:
  - a. Project board (also called steering committee): defines the project mandate (also called project charter) and takes major decisions such as the approval of deliverables and major changes in the project plan.
  - b. Project teams: taking care of specific tasks or management processes.
  - c. Typical roles are: software architect (mainly responsible for design), software engineer (mainly responsible for construction) and management roles (see management processes).

2. Project plan, consisting of phases with deliverables as output:
  - a. Each phase consists of tasks.
  - b. The set of phases has a partial order dependency based on the deliverables.
  - c. Each phase has a time window, possibly with intermediate milestones or deadlines. This is often represented in a Gantt chart.
  - d. Each task has resources, with specific roles assigned to it.
  - e. Each task has specified deliverables: process reports and artefacts.
3. Management processes, guaranteeing that the project proceeds in good order:
  - a. Resource management: time and budget.
  - b. Risk management: identifying risks and compensating measures.
  - c. Quality control: guaranteeing the quality of the artefacts and the engineering process.
  - d. Configuration management and change control: managing the versions of artefacts and the processes concerning the change of these artefacts.

## 5. Life cycle models

There are many different roadmaps to develop a system. They differ in the way the production of artefacts is assigned to tasks, the tasks are grouped in phases and the order in which the phases are executed. We do not prescribe a particular roadmap, but instead we give the characteristics of three of them:

- Sequential roadmap
- Evolutionary roadmap
- Incremental roadmap

In the sequential roadmap, also called the waterfall model, for almost each artefact one task is defined and the tasks are grouped in the way we have grouped the artefacts. So, in our terminology, there are 6 phases: inception, requirements, design, construction, testing and deployment. Note that “testing” here is only the testing of the programs and does not contain other verification activities. In the sequential roadmap the phases are executed one after another. The next phase may only start if the previous one has been successfully closed. If a problem occurs and it turns out that the cause of the problem is an error in one of the former phases, the project has to be “reset” to the task before the one with the error. Since errors often have only a local effect, it is not necessary to stop other tasks and therefore the sequential method is almost never executed in a purely sequential way: there is always some parallel execution in it and there is iteration if some error occurs. The other roadmaps are variations of the sequential roadmap.



In the evolutionary roadmap, also called iterative method, the phases before deployment, are executed several times in an iterative way. In each cycle the system is improved by solving problems discovered in a previous cycle. After a number of iterations the system is sufficiently good to move to the deployment phase. The advantage is the early feedback of the stakeholders, which avoids spending development effort in a wrong direction.

In the incremental roadmap the system is divided into parts and for these parts a sequential or evolutionary roadmap is performed, possibly in parallel. The parts need not be developed at the same time, but as soon a part has passed the testing phase, it is taken to the deployment phase, which means that parts of the system are already in use while other parts are still in an early phase. The advantage is that parts that are easier to develop or the parts that are most wanted, are already in operation while others are still under development.

## 6. Glossary

Words in italics occurring in the descriptions refer to other terms in the list.

### **actor**

A person or external-system that interacts directly with the *system*. The persons are called *users*.

### **architecture**

An architecture is:

- A set of *models* that describe the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, from various viewpoints.
- The principles guiding the design and evolution of the system.

### **basic software**

Software that is used by the computer hardware to give the system its basic functions, like an operating system or performs elementary tasks such as a compiler.

### **business case**

Description of the system in terms of the stakeholders that have to make the decision to develop the system, together with an analysis of the development and operational cost of the system, and of the benefits of the

system and the revenues it might generate.

### **component**

A recognized part of a *system*. Typically, a component provides cohesive functionality and has explicit context dependencies only. An OTS component, where OTS stands for “Off The Shelf”, is a component that already exists and that is not primarily constructed for the information system under development. Typically, OTS components are obtained from third parties.

### **extra-functional requirement**

A requirement considering a non-functional property of the system such as security, distribution, performance or reliability.

### **functional requirement**

A requirement that specifies a function that a system or component must be capable of performing. These requirements define behavior of the system, that is, the fundamental process or transformation that the information processing system or its components perform on inputs to produce outputs.

### **information processing system**

The system under construction, consisting of a *software system* and a *platform*. We call an information processing system often a *system* for short. We distinguish three types of information processing systems:

- Embedded system if the target system is a piece of *equipment*.
- Information system if the target is some *organization*.
- Basic system if the target system is a computer system itself (e.g. operating systems are compilers are examples of basic systems).

### **interaction**

The mutual influence of two actors, and/or components. Interaction is performed via an interface.

### **interface**

For two components, or a component and an external actor, a model of the types of the messages that are exchanged and the order in which this may occur.

### **model**

A formal representation of an aspect of a system. Typical examples are data models that give a static view and process models that give a

dynamic view.

**platform**

The part of the information processing system that is used to execute the *software system*. The platform normally consists of computer hardware and basic software.

**requirement**

A property that is demanded to be fulfilled by an information processing system. Often requirements are distinguished in functional and non-functional requirements. The functional requirements prescribe behavioral aspects, such as the way the system must interact with actors and responds to requests. Non-functional requirements regard constraints that are visible in the explicit behavior of the system, such as resource requirements.

**software engineering**

Software engineering is the development and maintenance of software by the systematic application of engineering techniques in the software domain.

**software system**

Part of the information processing system that will be realized in software.

**stakeholder**

An individual, team, or organization in the target system or its environment, with interest in the system. Examples are clients, suppliers, employees, managers, owners and system administrators. All stakeholders that have interaction with the system are *users*. One stakeholder has the role of the principal one who formally gives the assignment to develop the system.

**system**

A generic term for a group of interrelated, interdependent or interacting elements serving a collective purpose. Software systems, information processing systems and target systems are typical examples in the context of software engineering.

**system engineering**

The process of developing a system that must fulfill a certain purpose using the systematic application of engineering techniques, and of which software engineering is a part, provided the system has a software

subsystem.

**target system**

The system that will be supported by the information processing system to be developed.

**use case**

A description of a coherent piece of functionality in terms the stakeholders understand. Often a use case is expressed as a set of *use case scenarios*.

**use case scenario**

A sequence of actions to be performed in the interaction between the system and the actor, when a system is performing a use case.

**user**

Persons that interact with the information processing system. They are a subset of the *stakeholders*.

**validation**

Often this term is used as a synonym for *verification*. More specifically the term is used to verify if a system satisfies its requirements.

**verification**

All acts of reviewing, testing, checking, auditing or otherwise establishing whether artefacts conform to specifications. In most tasks, artefacts are produced that serves as specifications for other artefacts. In particular, if the specifications are requirements, the term *validation* is used.

**view**

A view is a model representing the system as seen by a stakeholder from a specific viewpoint. Typical viewpoints are structure, behavior, functionality, security, distribution, performance and reliability.

**References**

- [IEEE] IEEE. Standards Collection. Software Engineering Institute of Electrical and Electronic Engineers. 1994.
- [PMBOK] A Guide to the Project Management Body of Knowledge (PMBOK Guide) – 2000 edition. Project Management Institute, 2000.
- [PRINCE2] Managing Successful projects with PRINCE2 (PRINCE Guidance).

The Stationary Office Books, 2002. See also [www.prince2.com](http://www.prince2.com).