

# Part Five: IT Project Management

## 11. Software Life Cycle Models

Today it is recognised that any Computer software has its own life cycle. This life cycle probably starts with the initial idea and is certainly not finished when the software becomes into operation. Since the first software projects have been realised (see figure 4), different life cycle models have been proposed and effectively used here. While our understanding of the software lifecycle is briefly shown in figure 75, we will discuss the heritage and the currently adopted models as well.

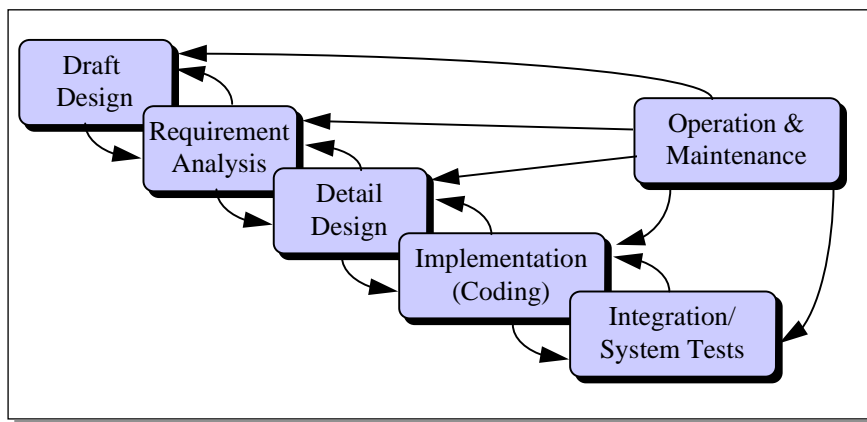


Figure 75: The main stages in the software life cycle [~ Petrasch1998]

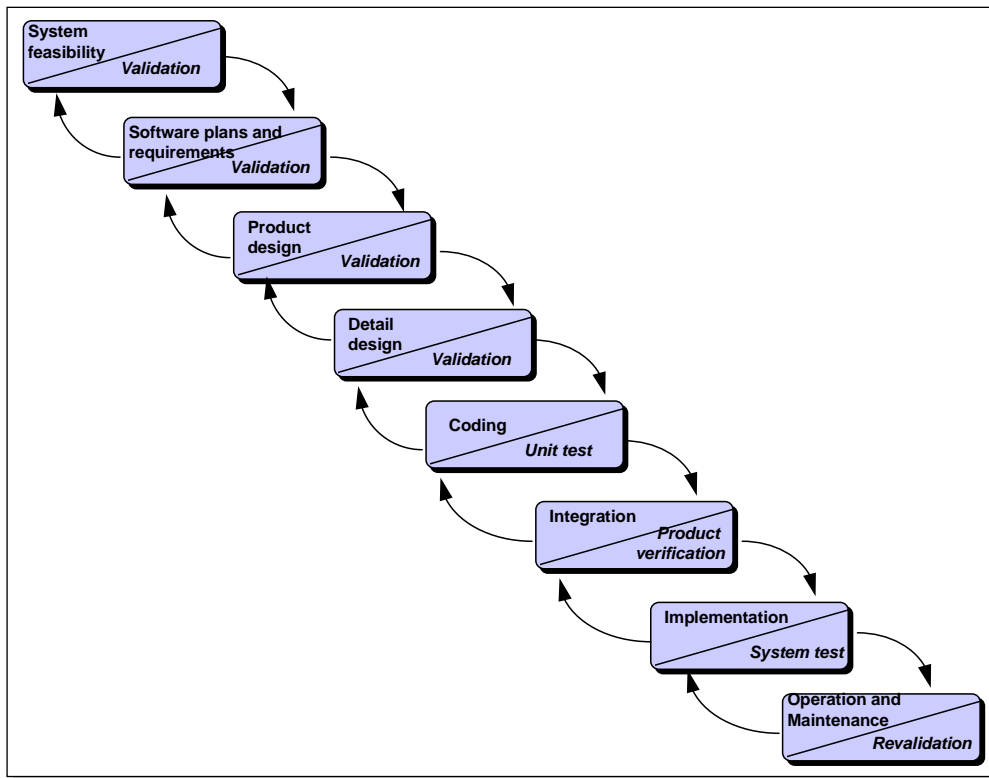
### 11.1 The Waterfall Model

The Waterfall Model is probably the earliest approach to improve software development, as it has been defined in 1956 already as a part of the Semi-Automated Ground Environment (SAGE) as so-called stage-wise model. The final description of the Waterfall Model was carried out in 1970 and is shown in figure 76.

The main features of the Waterfall Model are:

- Recognition of feedback loops between the different stages and thus the impact of errors on one stage is restricted to the next stage and not passed to the final product.
- The inclusion of prototyping in the software lifecycle as a so-called "build-twice" step.

As a result, the Waterfall Model demands an existing verification/validation process before entering the next stage. Since "open issues" are not accepted, in particular the development of interactive systems, requiring human intervention can not be handled efficiently, since the concept of "Use Cases" was not present yet.



*Figure 76: The Waterfall Model of the Software Live Cycle [Boehm1988]*

## 11.4 The V-Model

The V-Model is a process-oriented approach for software development as part of IT project management and has been inspired by Barry Boehm and further developed in Germany, in particular for Government and here in addition for military projects. The 1986 published V-Model is now superseded with its successor the V-Model XT.

Applying the V-Model for software developments can be seen as a cornerstone to comply with the ISO 9000 standards. It actually requires a particular infrastructure for software development and couples the responsibilities of Project Management, Quality Management, Configuration Management, and Software Development (figure 77).

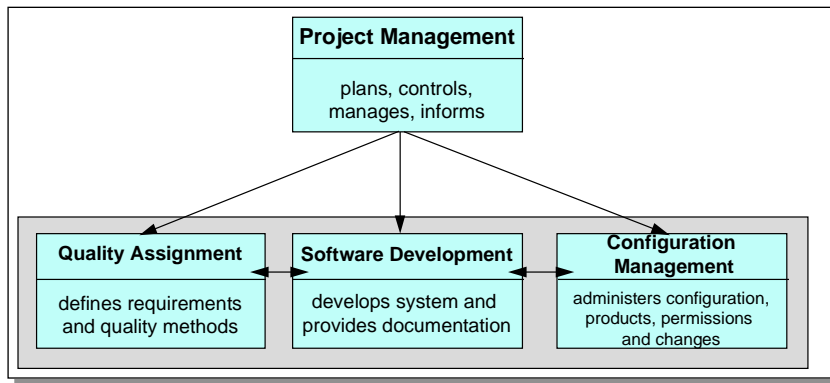


Figure 77: Interaction of the different PM levels according to the V-Model  
 [~ Petrasch1998]

Unlike classical project management approaches which defines phases and transitions on a time line, the V-Model is procedure-driven:

*The V-Model defines a set of inter-related Events and Activities for planning and design, software development, testing, and the final roll-out process.*

In addition, the V-Model is document driven:

*During project execution, the activities are broken down per management level and the resulting state has to be documented to provide the required transparency for the project evolution.*

Figure 78 depicts these dependencies and also provides a visual explanation for the term "V-Model (1997)".

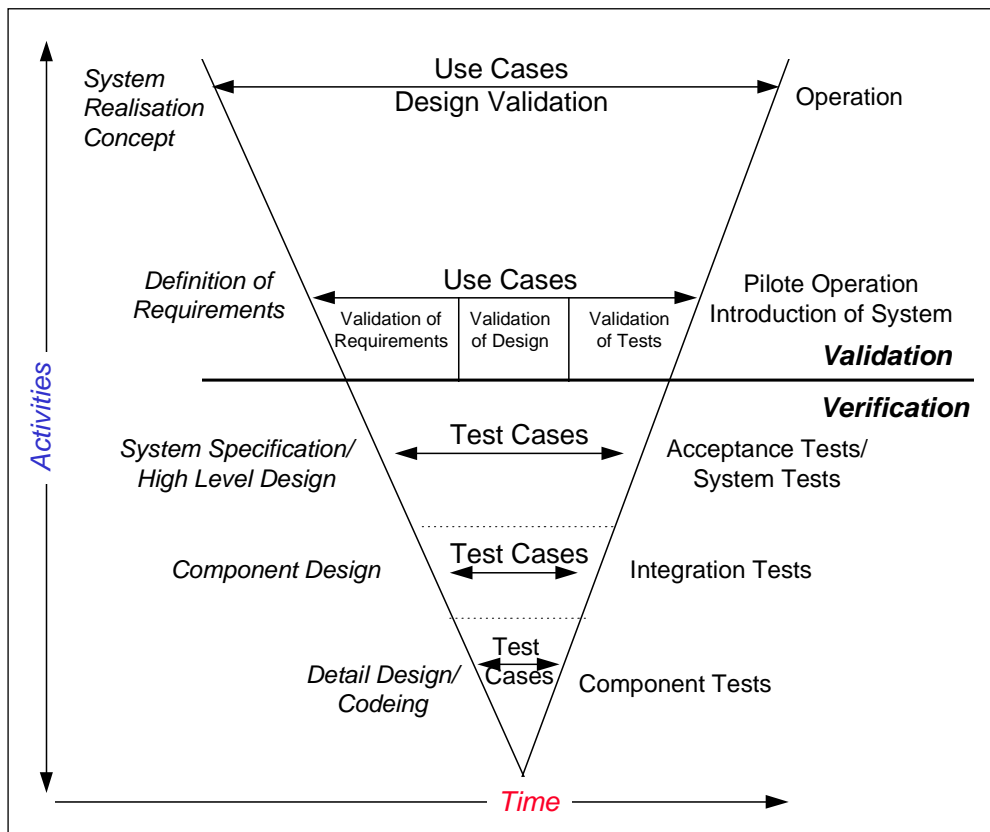


Figure 78: The V-Model

The drawbacks of the V-Model approach are the following:

- It formalises the whole project management and software development (and demands the respective documents) while burdening it with a lot of administrative tasks.
- Software development is strictly pre-determined, which is unrealistic; and any change has to be approved.
- It provides too little interaction among the team members and thus does not exploit the team creative capacity.

## 11.4 The V-Model XT

Both the V-Model 1997 and the new V-Model XT is maintained by the German BMI-KBSt (Koordinierungs- und Beratungsstelle der Bundesregierung) and made public available (in a set of documents downloadable). Within the V-Model XT several new approaches are incorporated which align with the achieved progress in the IT world:

- On the lowest level it introduces the Project Subject:
  - This might be a hardware solution,
  - a *software system*,
  - a *complex system* consisting of both,
  - an *embedded system*,
  - or perhaps *system integration*;
  - however it *does not* cover *system services* (which are out of scope).
- The definition of the Project Rôle tells who is using V-Model XT for coordination:
  - the customer (Auftraggeber) managing one or several the vendors (Auftragnehmer),
  - the vendor (Auftragnehmer) employing it itself or other sub-vendors,
  - both customer and vendor using the V-Model XT.
- Thus we achieve the definition of the Project Type, whether it is
  - a customer-driven project,
  - a vendor-driven project,
  - or a shared project where both parties using the V-Model XT.

Thus, the V-Model XT adjusts explicitly to the typical situation for Government-related development, where the Government acts as 'client' and the 'vendor' is an external company; chosen after the bidding process.

*The V-Model XT can be characterised as an incremental, adaptable, and model-driven procedure model.*

*Unlike the original V-Model it does not employ formal documents, but rather CASE tools are now considered for description and follow up.*

While the incremental structure of the V-Model XT is displayed in figure 79, it covers the main modules:

- PM: Project Management

- QA: Quality Assignment
- CM: Configuration Management
- PM: Problem- and Change Management

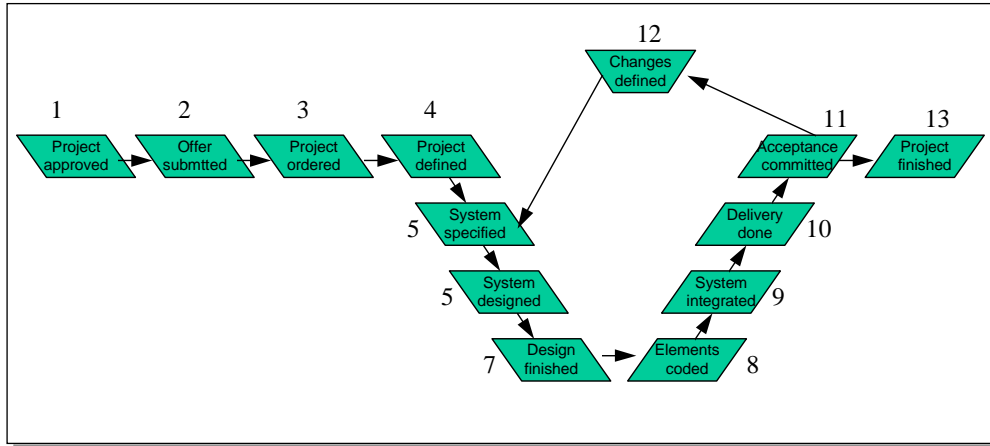


Figure 79: Incremental system development according to the V-Model XT (on the vendor side) [Bartelt, Ternité, Zieger2005]

Any change of the *Product* (or a sub-product) is called an *Activity* and has to obey the following phase changes:

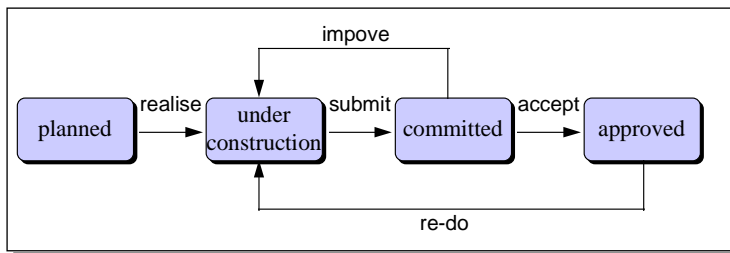


Figure 80: Phase changes of a product or sub-product

While the V-Model XT is mostly deployed in Germany it is recognised and used internationally, for instance for the US American ITS (Intelligent Transport System) and documented in the "System Engineering Guidebook For ITS":

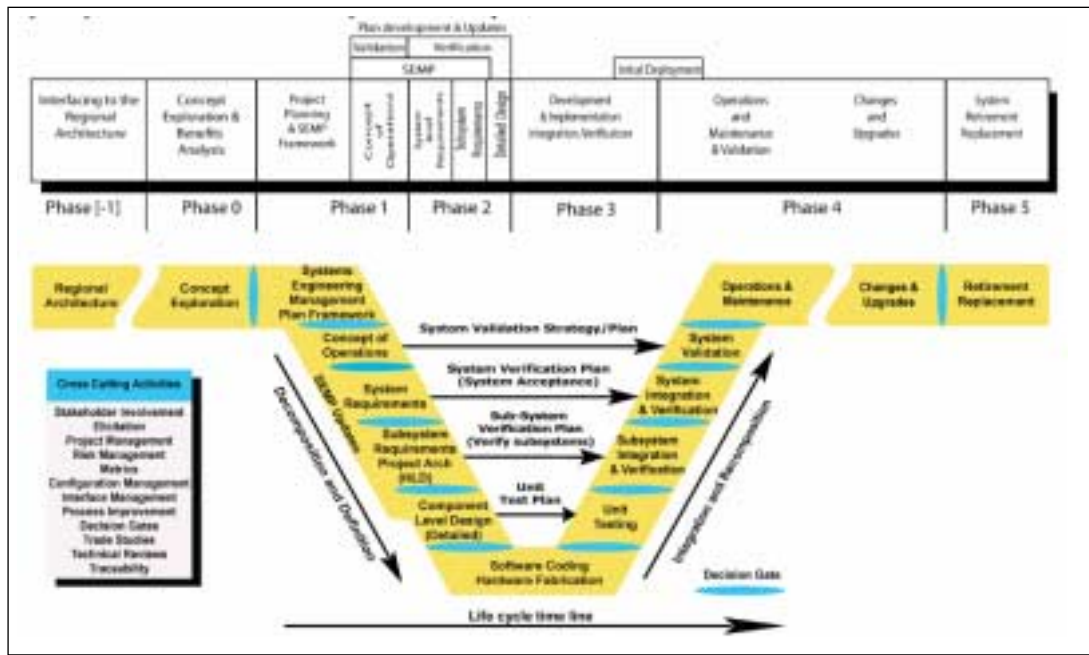


Figure 81: Adoption of the V-Model XT for the ITS [ITS2007]

## 11.5 The Spiral Model

The original Spiral Model was invented in 1988 by Barry W. Boehm and it projects the life-cycle of a (software) product on a (repetitious) spiral while defining four quadrants Q1 - Q4:

- Q1: Determines objectives, considers alternatives and constraints
- Q2: Evaluation of alternatives, identify and resolve risks
- Q3: Develop and verify, even consider next-level product
- Q4: Plan next phases

*Unlike the other models, the Spiral Model is risk-driven; after each "round" the risk to start with the next round is assessed.*

The risk assessment is based on certain key questions needed to be answered and documented:

1. What are the objectives?
2. What are the constraints?
3. What are the alternatives?
4. Which risks are involved?
5. How to resolve the risks?
6. What would be the result of the risk resolution?
7. What are the plans for the next phase?
8. What are the commitments?

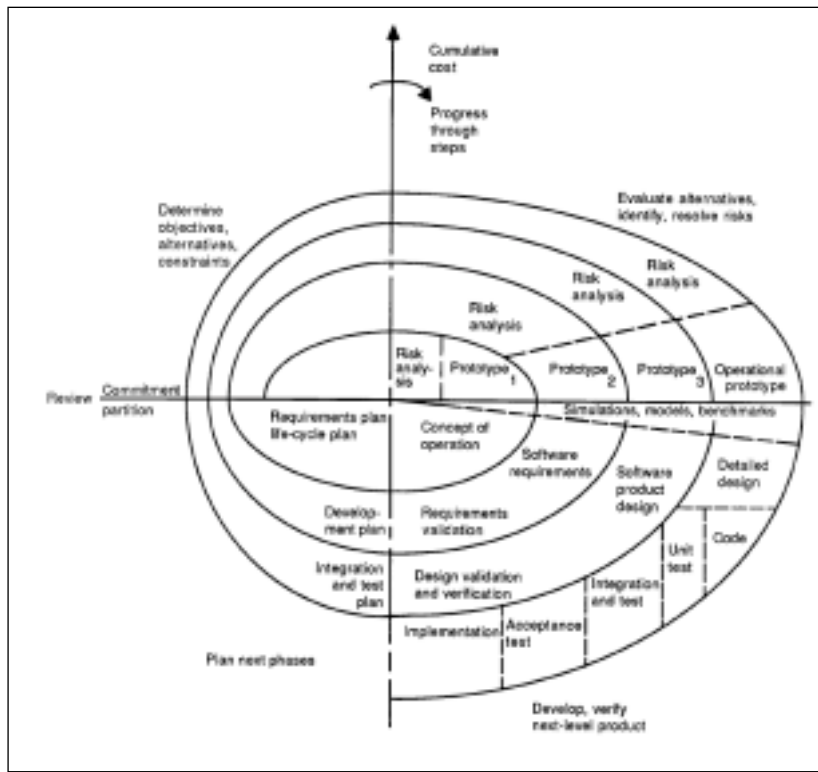


Figure 82: Spiral Model according to the Boehm [Boehm1988]

In this context, each cycle ("a round") can be considered as stage (figure 82) :

- Round 0: Feasibility study
- Round 1: Concept of operation
- Round 2: Specification of the top-level requirements

*Note: The Spiral Model introduced in chapter 7.2 (figure 32) is a milestone-driven approach and thus differs in its scope from Boehm's approach.*

## 11.7 GQM

GQM is the acronym for *Goal/Question/Metric* and has been introduced by Victor Basili at the University of Maryland in 1983 and improved by Dieter Rombach 1988. It can be seen as enhancements of the following :

- QFD Quality Function Deployment (developed by Yoji Akao 1966 at the Tamagawa University in Tokyo)
- SQM Software Quality Metrics (developed in 1980 by Marine for Metrics Incorporated)

approaches and is suitable for Quality Management and Project Management as well.

*GQM is a paradigma-driven approach; the result of any measurement is only valid in the context, as provided by the GQM plan*

Within the GQM approach, the following phases are considered (figure 83):

1. The GQM plan
2. The definition of the questions and the metric for measurements
3. The collection of the measurement data
4. The interpretation of the measurements in the given context

Unlike other models, it respects the origin of the measurement data; sensible data are sheltered against random access.

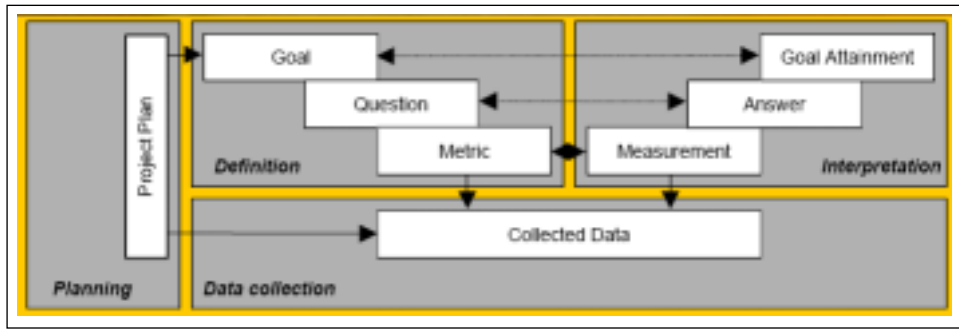


Figure 83: The four GQM phases [Bernard2005]

Most important for software development is, that GQM provides a procedural language allowing a modelling of its dependencies as shown figure 84. For Eclipse, a plug-in FOCUS is available which allows the modelling.

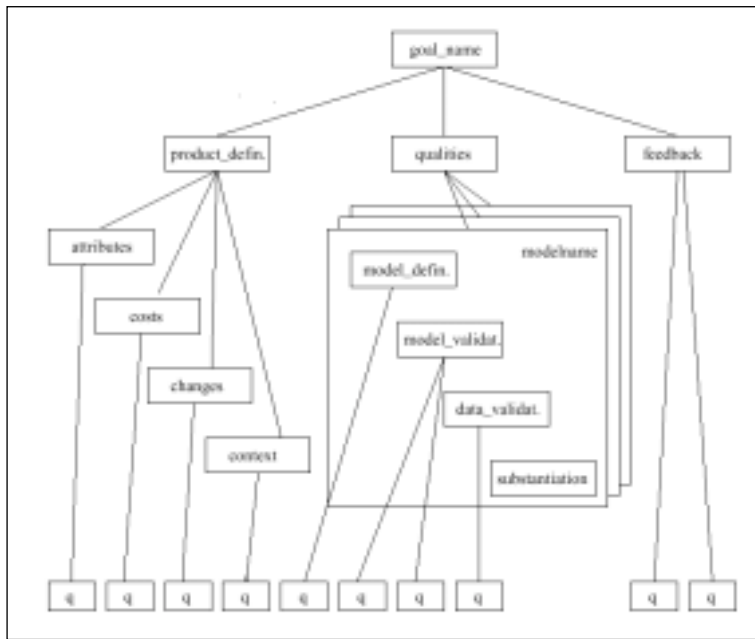


Figure 83: Schematic lay-out of a GQM plan [Differding1993]

## 11.8 The RUP Model

The RUP Model has been created by the company Rational (like Rose, Clear Case and other Rational software products) in 1996, which has been taken over by IBM.



RUP is as procedure model for software development and currently available in version 9 (figure 84).

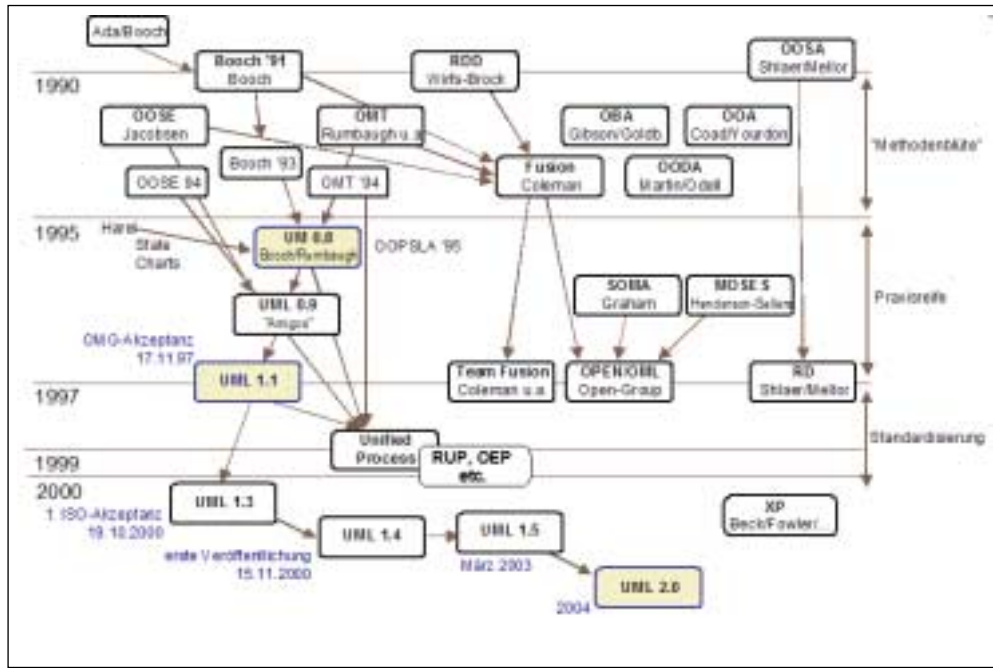


Figure 84: Development of OO modelling frameworks [Zockoll2004]

Like Rose, RUP uses an Object Oriented approach for software development and basically divides software development into two independent (orthogonal) streams:

- *Disciplines:*
  - Business Modelling
  - Requirements
  - Analysis & Design
  - Implementation
  - Test
  - Deployment -- and additionally
  - Configuration & Change Management
  - Project Management
  - Environment
- *Phases:*
  - Inception (initial design providing the *Lifecycle Objectives LCO*)
  - Elaboration (design, yielding the *Lifecycle Architecture LCA*)
  - Construction (coding, introducing the *Initial Operational Capabilities IOC*)
  - Transition (phase-change, responsible for *Product Release RP*)

*In this respect, RUP is an iterative-incremental approach to software modelling.*

Figure 85 demonstrates which disciplines are relevant to what extent in the different phases. For the different phases RUP defines sets of activities to be carried out.

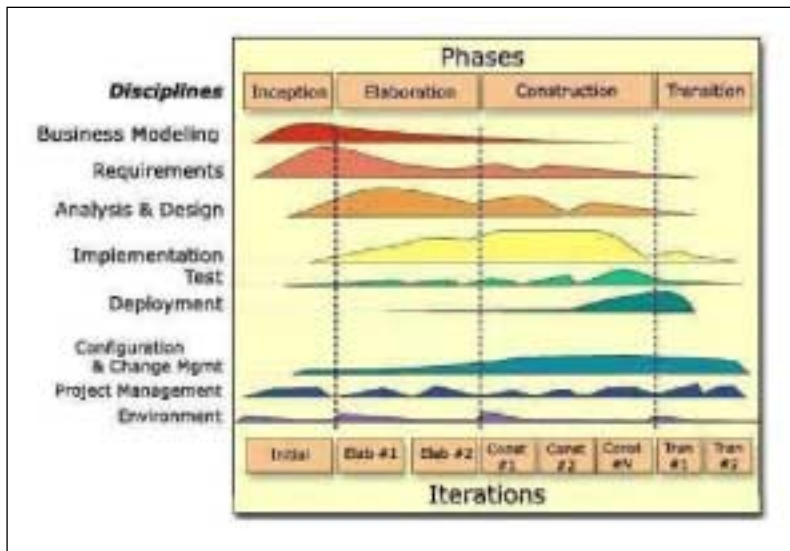


Figure 85: Intensity of the different RUP disciplines over the phases [Ambler2005]

## 12. Software Modelling

### 12.1 Software Metrics

Software Metrics is the approach, to define a measure for the given code (sizing). The measure is typically an one-dimensional value based on a certain method. The following methods are known:

- Number of Lines of Code (LoC), expressed as kLoC or MLoC.
- Number of words, characters files.
- Number of functions or subroutines.
- Number of classes (for OO languages).

Which measure we use, depends on the programming language. The measure itself has to be unified, thus the same formula to determine the size has to be used. The rationale behind that measure is to correlate its value with the behavioristic behaviour of software development:

- Law of continuous change: Once a Use Case is subject for a change, all relevant software modules have to be changed.
- Law of growing entropy: During its evolution, the code becomes less and less maintainable.
- Law of a statistical smooth growth: A typical software project will produce a continuous and smooth growth of the code base.
- Pareto's law claims, that 20% of the code impacts 80% of the result.

- The law of the re-usability requires: "*Don't solve any problem that has already been solved.*" -- check your software repository for already existing solutions unlike develop them again.
- The Parkinson's law includes:
  - "Work expands to fill the available time."
  - "There's always time to do the *right* projects, but there's never time to do them over."
  - "Adding manpower to a late software project makes it later."

The most common approach for the software metric is to use the LoC. Figure 86 shows the dependency regarding the development of Windows NT.

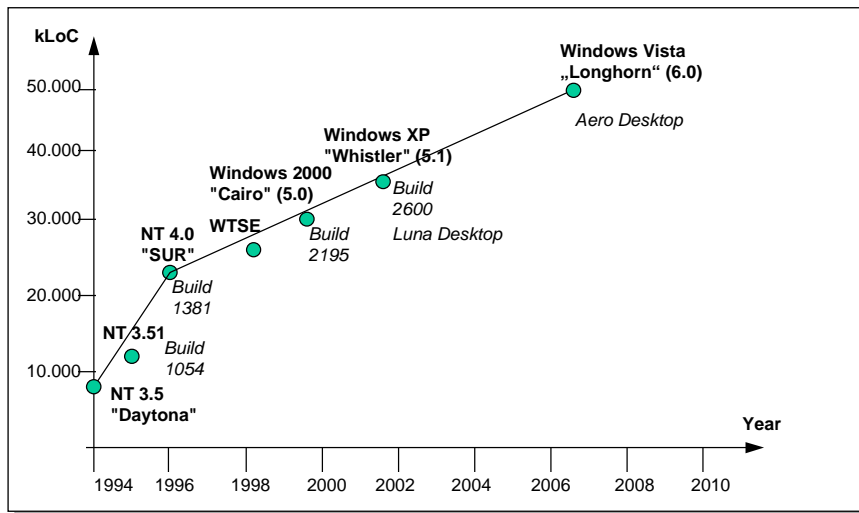


Figure 86: Code enhancement for the Windows NT product family

Table 7 provides a breakdown of the metrics for some E-Mail servers systems under UNIX. Here, we can see, that these different measures provide an estimate of the modularity of the systems.

Mail Transfer Agent	Lines	Words	Characters	Files
qmail-1.03	16.617	44.780	395.243	279
sendmail-8.9.1	55.059	179.376	1.229.121	54
zmailer-2.2e10	57.595	205.524	1.423.624	227
smail-3.2	62.331	246.140	1.701.112	151
exim-2.02	70.102	283.295	2.172.786	128

Table 7: Metrics for some UNIX Mail Transfer Agents

## 12.2 Software Modelling and CASE Tools

In software design several distinct design documents are relevant:

- *High Level Design document (HLD)*, describing the products functionality in a bird's view, and the
- *Low Level Design documents (LLD)*, detailing the individual software components (and thus are also called *Detailed Design* documents).

In particular in Germany, the HLD is fed from the following input documents:

- *Pflichtenheft* - the agreed specifications between the customer and the software vendor/developer.
- *Lastenheft* - the requirement specification as provided by the customer.

The descriptions herein follow a formal modelling language and are provided in terms of *charts*. In order to produce the respective charts and to make them consistent for a given software layout. *CASE* tools (*Computer Aided Software Engineering*) are used in the design process.

Today, three different modelling languages are common:

- UML - Unified Modelling Language
- ERM - Entity Relationship Model
- SA/SD - Structured Analysis/Structured Design

The CASE tools allow during the modelling process the determination of the dependencies of the software components. Thus, while deriving the WBS structure, CASE tools can be efficiently used to support this evaluation. Figure 87 shows a sample, how to model with the *Eclipse* plug-in *Together*.

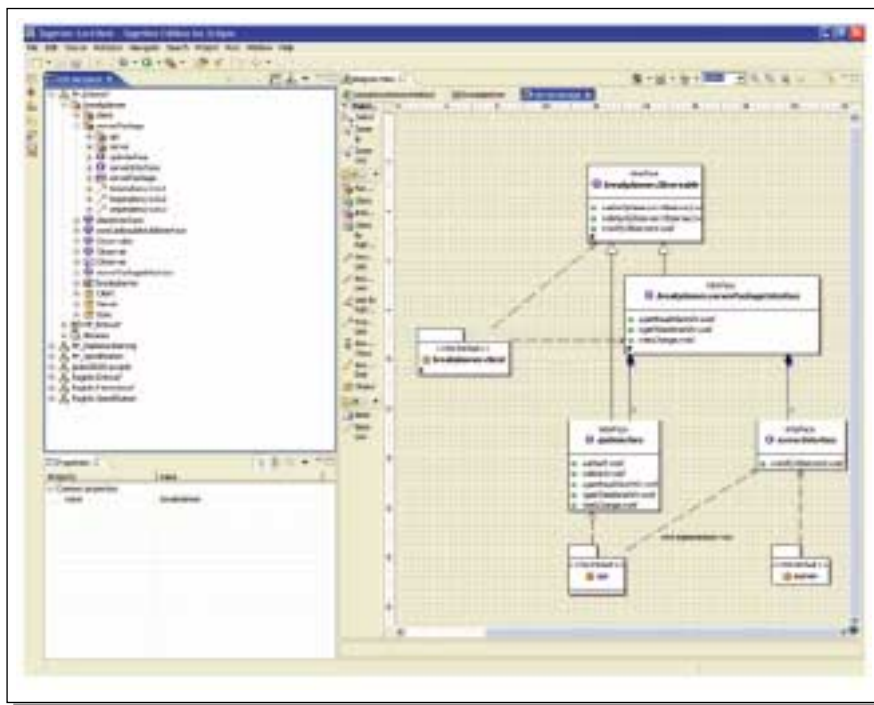


Figure 87: CASE modelling with Together [Bartelt, Ternité, Zieger2005]

## References

[Ambler2005] Scott W. Ambler; A Manager's Introduction to Rational Unified Process (RUP)

[Boehm1988] Barry W. Boehm; A Spiral Model of Software Development and Enhancement

[Bartelt,Ternité,Zieger2005] Christian Bartelt, Thomas Ternité, Matthias Zieger; Modellbasierte Entwicklung mit dem V-Modell XT, [www.sigs-datacom.de/2005](http://www.sigs-datacom.de/2005)

[ITS2007] System Engineering Guidebook For ITS, Verison 2.0, 2007

[Zockoll2004] Wikipedia/RUP

[Differding1993] Christiane Differding; Ein Objektmodell zur Unterstützung des GQM-Paradigmas, Diplomarbeit Universität Kaiserslautern 1993

[Bernard2005] Yvonne Bernard; Seminar Erfahrungen und Experiment im Software Engineering, Universität Hannover, 2005

## Further Reading

<http://www-pu.informatik.uni-tuebingen.de/users/klaeren/sw.pdf>