

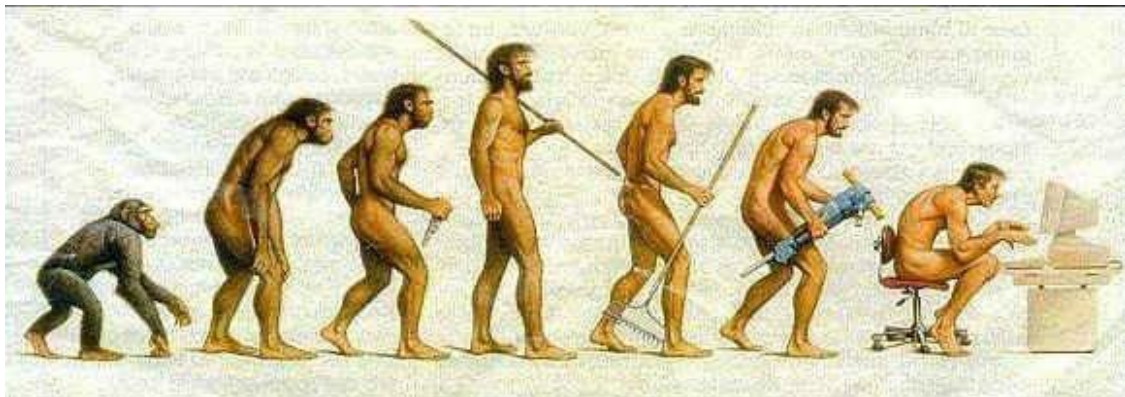


SERIOUS

DELIVERABLE D2.2

Best practices of Evolutionary SW Development

.....



Project number: ITEA 04032
Document version no.: WP2 Deliverable 2.2, Final Version
Edited by: PHI / CAL

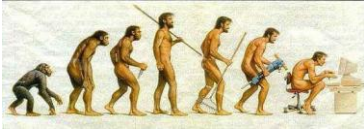
ITEA Roadmap domains:

Major: Services & software creation

ITEA Roadmap categories:

Major: Software engineering

Minor: System engineering



HISTORY

Document version #	Date	Remarks
V0.1	23-3-2007	Draft chapter 1 and 2
V0.2	24-4-2008	Second draft all chapters
V0.3	05-05-2008	Added process pattern look-up table
V0.4	06-06-2008	1 st review
V0.5	15-06-2008	Incorporated review comments
V0.6	20-06-2008	2 nd review
V0.7	07-07-2008	Incorporated all review remarks
V1.0	12-08-2008	Final Version
V1.1	29-08-2008	Final Version, some textual consistency changes

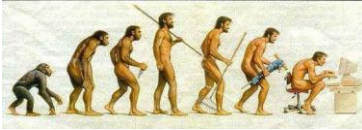
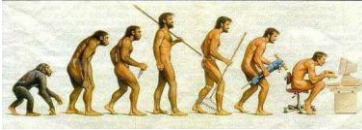
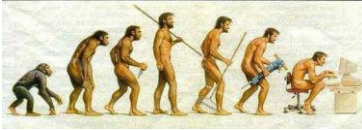


TABLE OF CONTENTS

1	EXECUTIVE SUMMARY	5
2	CONCEPT OF PROCESS PATTERNS.....	6
2.1	Evolutionary Development Process Patterns.....	6
2.2	Describing Evolutionary Process Patterns.....	6
2.3	Process Pattern template.....	6
3	OVERVIEW OF PROCESS PATTERNS	9
4	REQUIREMENTS RELATED PROCESS PATTERNS.....	11
4.1	Rapid UI prototyping pattern	11
4.2	Requirement Impact Description (RID) pattern.....	12
4.3	Feature description pattern	19
4.4	Light analyses of a Feature pattern.....	21
4.5	Delta Specifications pattern.....	23
4.6	Manageable requirements traceability	31
5	DESIGN RELATED PROCESS PATTERNS.....	38
5.1	Critical Computer Resource Management pattern	38
5.2	Multidisciplinary product configuration management pattern	41
5.3	Software FMEA pattern	44
5.4	Security and Privacy pattern.....	49
6	CODE RELATED REALIZATION PROCESS PATTERNS	54
6.1	Continuous Builds Pattern.....	54
7	TESTING RELATED PROCESS PATTERNS	56
7.1	Risk based Testing pattern	56



7.2	Incremental Testing pattern	60
7.3	Technical Review pattern	64
7.4	Verification & integration in incremental development pattern.....	67
8	SUPPORTING PROCESS RELATED PATTERNS.....	72
8.1	Incubators for reducing project risk pattern.....	72
8.2	Incremental Configuration Management pattern	75
8.3	Defect Rootcause Analyses pattern	80
8.4	Proactive Quality Assurance pattern	87
8.5	Quality Assurance driven Process Improvements pattern.....	91
8.6	Estimation in evolutionary SW development pattern.....	97
8.7	Baseline auditing and configuration status accounting pattern	106
8.8	Software Development Stream pattern	112
8.9	Product baseline overview pattern.....	118
9	REFERENCES.....	123



1 Executive Summary

The SERIOUS project focuses on maximizing the long-term value of an investment made in software development by utilizing evolutionary software development models aimed at prolonging the lifetime and improving the quality of a software product. This is done by taking into account the requirements of the whole lifecycle of the product from the beginning of the development.

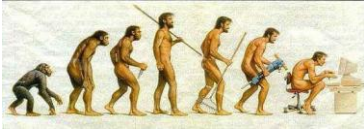
Most software development models focus extensively on the initial development phase, not taking into account the requirements for the period from the delivery to the end of the product's lifecycle. Evolutionary software development models, as defined in the SERIOUS project, are methods that are aimed at combating software quality degradation, usually beginning right after the initial development cycle has ended.

The focus of the SERIOUS project is on practical methods that are applicable in real-life development of software intensive systems, most of which have been in production long before the project and which may still last several years or decades before being replaced by whole new systems. It is therefore important that the evolutionary development models can be applied to improve the quality of both new systems to be developed from scratch and existing systems already in production.

Upon investigation of the development models that are currently available, within both industry and academia, it became obvious that none of the current models were able to meet the goals of the project. Although the so-called agile software development models generally allow for the flexibility required for software to evolve during its lifecycle, thus contributing to the quality of the software even after its initial release, even they did not meet the criteria required to be effectively applied to the ongoing software development processes.

The available models did not take into account the current trend of distributed software development crossing organizational boundaries or the increasing complexity caused by more and more software components interacting and integrating with each other.

In order to achieve practical results, it was therefore decided that instead of trying to identify a single model which would meet the goals of the SERIOUS project, it would be better to gather a set of tried and tested best practices which have been successfully applied in real life software development projects from the Serious partners.



2 Concept of Process Patterns

The Industries Best Practices in this document are described as Evolutionary Software Development Process Patterns, further referred as Process Patterns.

A Process Patterns is a universal way to describe re-usable best practices of a development process. It is developed in analogy to Object Oriented Design patterns.

A Process Pattern is a standard (process) modular solution for implementing evolutionary development processes.

2.1 Evolutionary Development Process Patterns

As part of the SERIOUS project we have developed a notion of process patterns in the context of evolutionary development. The main idea behind these process patterns is to stimulate exchange of knowledge between different industries. Practices are gathered and more or less isolated in order to facilitate the incorporation of the process pattern in another development process (without having to change too much in the existing process or organization). Changing organizational processes is maybe the hardest thing to achieve. We hope to realize fast and clear results by applying evolutionary process patterns in an organization.

2.2 Describing Evolutionary Process Patterns

How do we describe evolutionary process patterns? Currently used graphical notations, as in Object Oriented Design Patterns, while important and useful, aren't sufficient. They simply capture the end product of the design process as relationships between classes and objects. To reuse the evolutionary process, we must also record the decisions, alternatives, and trade-offs that led to it. Concrete, proven examples are important too, because they help you see the design in action.

In the Serious project, we have developed a template for describing Process Patterns. The template lends a uniform structure to the information, making process patterns easier to learn, compare, and use.

In the next paragraph, the layout of this template is given

With respect to the size of a process pattern, the following rule of thumb applies: It should not contain more than 4 to 5 pages. If more pages are needed to describe the Process Pattern, the subject being covered maybe too large / less modular. In that case it is advised to reconsider the scope of the Process Pattern.

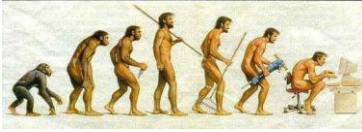
2.3 Process Pattern template

2.3.1 Pattern Name and Classification

- Pattern Name and Classification (Guideline: Couple of words that conveys the essence of the pattern and that becomes part of the vocabulary)

2.3.2 Intent

- Some sentences describing the intent of the pattern
- What does the pattern do?
- What is its rationale and intent?



- What issue or problem does it address?

2.3.3 Also Known As

- Other names of the pattern

2.3.4 Motivation

- A scenario that illustrates the problem (from experiences)
- How the pattern solves the problem
- (Guideline: 1/2 page)

2.3.5 Applicability

- What are the situations in which the pattern can be applied? E.g. in which kind of process could it fit, in which it will certainly not fit (in terms of e.g. used process, organization, particular product aspects).
- What are examples of poor performance that the pattern can address?
- How to measure whether it is successful?
- How can you recognize these situations?
- (Guideline: Couple of bullets)

2.3.6 Structure

- A graphical presentation of the pattern, or some plain text
- Maybe UML picture or flow diagram

2.3.7 Participants

- Roles/People?
- Other processes (e.g. outside development)?
- Explain which people and other processes play a role in the pattern
(Guideline: 1/2 page list)

2.3.8 Collaborations

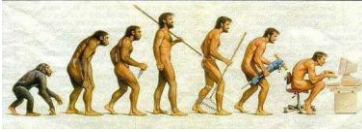
- How the participants collaborate to carry out their responsibilities
- (Guideline: 1/2 page)

2.3.9 Consequences

- How does the pattern support its objectives?
- What are the trades-off and results of using the pattern?
- (Guideline: 1/2 page)

2.3.10 Implementation

- What pitfalls, hints, or techniques should you be aware of when implementing the pattern?
- (Guideline: 1/2 page)



2.3.11 Sample

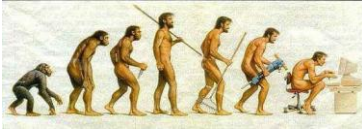
- Provide fragments that illustrate how you might implement the pattern. Typically provide examples of documents, procedures etc.
- (Guideline: 1 page)

2.3.12 Known Uses

- Examples of the pattern found in real systems. Give for instance examples of organizations that applied the process pattern and provide for instance some statistics about the frequency of use. Also experiences when someone introduced the process pattern in an existing (other) development process is important.
- (Guideline: 1/2 page)

2.3.13 Related Patterns

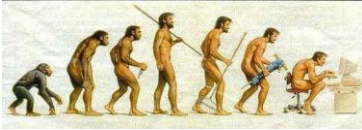
- What patterns are closely related to this one?
- What are important differences?
- With which other patterns should this one be used?
- (Guideline: Few sentences per related pattern)



3 Overview of Process patterns

Overview available Process Patterns

Pattern	Keywords
Requirements related patterns	
Rapid UI prototyping	Rapid UI prototyping method replaces early written software specifications with a “fully working” HTML prototype. Agile method,
Requirements Impact description	Requirement Impact Description (RID), PRES (Project Requirements Sheet)
Feature description	high-level system requirements, customer requirements
Light Analysis of a feature pattern	First Analysis, Initial Analysis, Initial assessment of a feature, Budgets estimates
Delta SRS & System overview (incremental requirements documentation)	Requirement Impact Description (RID), requirements' gathering, re-used components, adding isolated features, System Overview, Delta System Requirement Specification, impact on existing requirements and design, define incremental requirements, evolutionary product development
Manageable requirements traceability.	requirements traceability, Bidirectional requirements traceability , Traceability Matrix, Unit Verification
Design related patterns	
Critical Computer Resource Management	CPU memory, CPU capacity, disk space, object oriented languages, Field Programmable Gate Arrays, On Board Controllers, Network Processors, CCR requirements, CCR estimates, Capability Maturity Model (CMM), Capability Maturity Model Integration (CMMI)
Multidisciplinary product configuration management	Multiple view points, Conceptual Product Structure, Commercial Product Structure, Engineering Product Structure, Manufacturing Product Structure, Customer Support Product Structure
Software failure mode and effect analysis	Increase product quality and reliability, identification of failures, pre-design step
Security and Privacy Pattern	Personal data, intellectual property, security/privacy requirements, Product Security Leadership Council, OCTAVE
Code related realization patterns	
Continuous Builds	software builds, quality of the software, effectiveness of the development, critical errors, agile method
Testing related patterns	
Risk Based Testing Pattern	Product Risk Analysis, total costs of defects, Product Test Risk Matrix
Incremental testing-Master Integration Diagram	“integration is leading” approach
Technical Review	Methodic way to assess the quality of your



deliverables

Verification & integration
in incremental
development

Integration plan (IP), Master Test Plan (MTP), System
Verification Specification (SVS), Supported
Configurations Specification (SCS), V-model, Fade in /
Fade out operations, pre production models,
Configuration control, Integration / Testing,

Supporting process related patterns

Incubator for reducing
project risk

reducing project risk, lifecycle development, Pre-
proposal and proposal phases, Validation/review
phase, project launching, project establishment,
configuration management aspects, Software
Components, configuration items, product stability &
quality, risk reduction, timing constraints, Quality levels
for a CI, product archive remains stable

Incremental
Configuration
Management

Determine the weaknesses, problem reports, Fight the
cause, Learn from the mistake, Prevent defects,
Safety based root cause analysis, Production based
root cause analysis, Process based root cause
analysis, System based root cause analysis, Data
collection, Pareto, Four-blocker" sheet, Phase related
root causes, Human related root causes, Fish-bone
diagram(Ishikawa diagram)

Defect root cause
analysis

Proactive Quality
Assurance

Process Quality Assurance, Product Quality
Assurance, Project Quality Assurance, Quality
Management

Quality Assurance
driven process
improvements

Process/Product/Project/Software/Development
Quality Assurance, track the non-compliances to
closure, non-compliances, Plan Do Check Act
(PDCA).

Effort estimation in
evolutionary SW
development

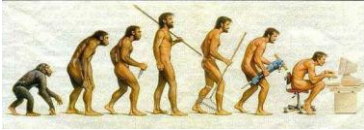
Analogy Method, Fuzzy Logic, Matrix Sizing Method,
Standard Component Sizing Method, Change Sizing,
Wideband Delphi Technique, Selecting size metrics

Baseline auditing and
configuration status
accounting
Software development
stream pattern

Configuration auditing, Status accounting,
Configuration Management Plan, Analyzing build
results, Maturity grid
Multiple development, Evolutionary development,
Multi-site development, Development of an embedded
system, Cascade model

Product baseline
overview

Configuration management at the product level,
Configuration Items



4 Requirements related Process Patterns

4.1 Rapid UI prototyping pattern

4.1.1 Pattern Name and Classification

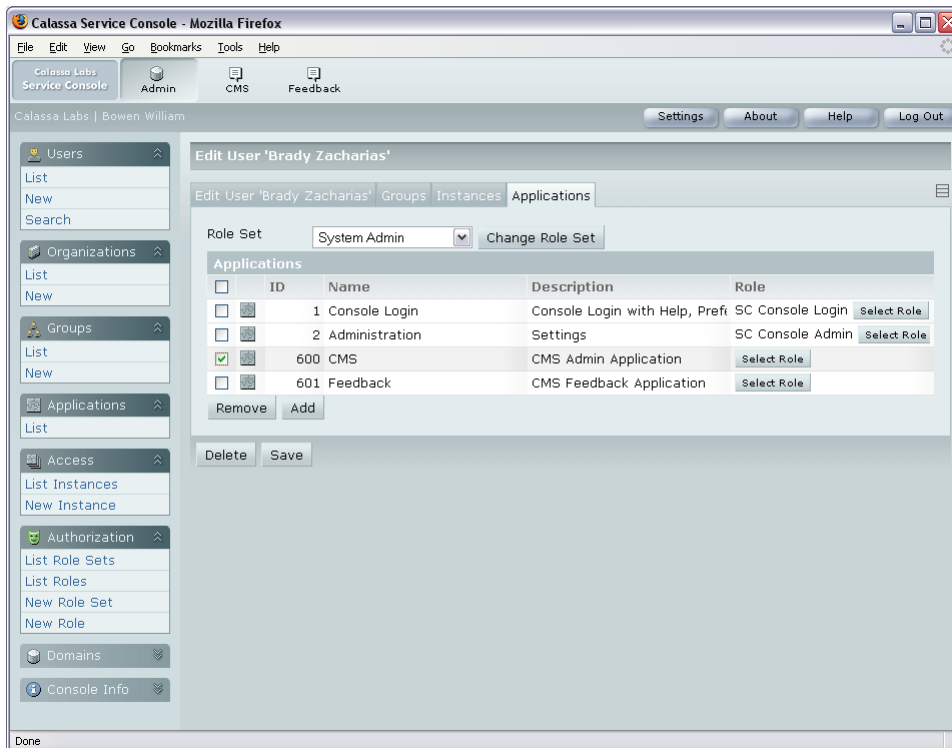
The **Rapid UI prototyping** method replaces early written software specifications with a “fully working” HTML prototype. It enables the software architect to visualize the interactions and potential logical problems in the software from early on.

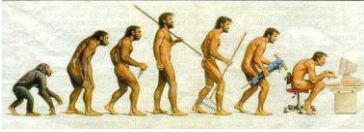
4.1.2 Intent

Whilst rapid UI prototyping is a powerful tool to communicate and sell the software internally and externally, it also enables rapid software design by reducing the amount of initial written documents and the time spent on updating the documents. Rapid UI prototyping can produce the final user interface as a side product of the specification phase.

4.1.3 Motivation

The initial software specification phase takes a lot of time and resources with no guarantee of the applicability of the initial design documents. Replacing the early written design documents with rapid prototyping accelerates the software development and enhances the quality of the software via better internal and external communications.





A picture is worth a thousand words, a “working” UI prototype is worth more than a thousand words.

4.1.4 Applicability

The usefulness of the method increases when the complexity of the software increases. Also, the benefits increase as the number of people/organizations involved in the development process increases. The method is not suitable for software with no user interface. It also does not benefit simple design processes.

4.1.5 Participants

The participants include software architects, software developers, UI designers, product manager, marketing, etc. personnel, and internal/external customers.

4.1.6 Collaborations

Once the customer together with marketing and product manager have decided on the functionality, the software architect and UI designer produce the prototype which will then be distributed to all parties involved for further actions.

4.1.7 Consequences

Rapid UI prototyping speeds up the development process and eases the communications between the participants. Being an agile method, it highlights potential problems in the design and enables to address them early on. It should be noted, that the method requires a competent UI designer to work with the software architect.

4.1.8 Implementation

The method may lead to overoptimistic expectations from the customer. They may feel they have seen the finalized product when they actually have seen the prototype. The method may be difficult to implement in organizations with narrow roles/competences.

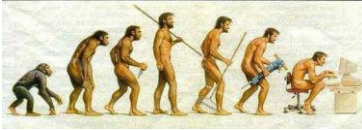
4.1.9 Known Uses

Although there most probably are other organizations utilizing a similar approach to software development, Calassa Labs is the only known user of the rapid UI prototyping method.

4.2 Requirement Impact Description (RID) pattern

4.2.1 Pattern Name and Classification

Requirement Impact Description helps identifying the impact of several stakeholders requirements on the scope of the project.



4.2.2 Intent

Describe a high level requirement in an early stage of development focusing on its impact on existing requirements and design and provide a rough estimate of the effort.

4.2.3 Also Known As

PRES (Project Requirements Sheet)

4.2.4 Motivation

Determine the impact on requirements and design of adding isolated features to the system without the overhead of updating and formalizing large documents. This speeds up the initial phases of the project because it supports the discussion and decision making process regarding the contents of the projects instead of just doing paperwork.

Writing and updating the Requirements Specification for an entire system is often a lot of work when the system is big and complex. Especially in organizations where these changes occur frequent. Every update big or small of a System Requirements Specification brings overhead with it for reasons of configuration management. When changes are frequent the amount of overhead becomes un-workable. Sometimes requirements are scattered over many different documents which makes it even more complex to handle.

Changes also have to be explicitly recognizable among lots of other texts in the System Requirements Specification(s) because the changes may have to be removed again when the feature is removed from the project.

When a feature is added changes have to be done at several places in the SRS. When it is decided that the feature does not fit in the project and has to be removed then at all the places in the SRS the applicable text would have to be removed. It might be the case however that the text in the requirements (or design) has to stay because of some other feature that needs the change in requirements or design. When this should be tracked in one requirements or design it would be difficult to keep track of the necessary changes.

The following drawing gives a graphical presentation of this situation:

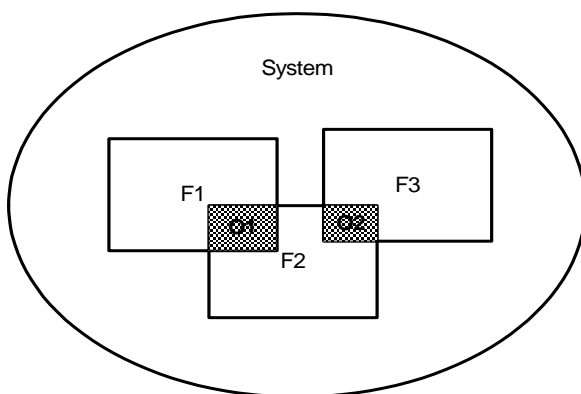
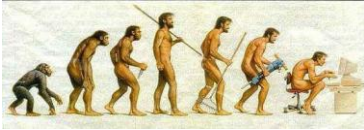
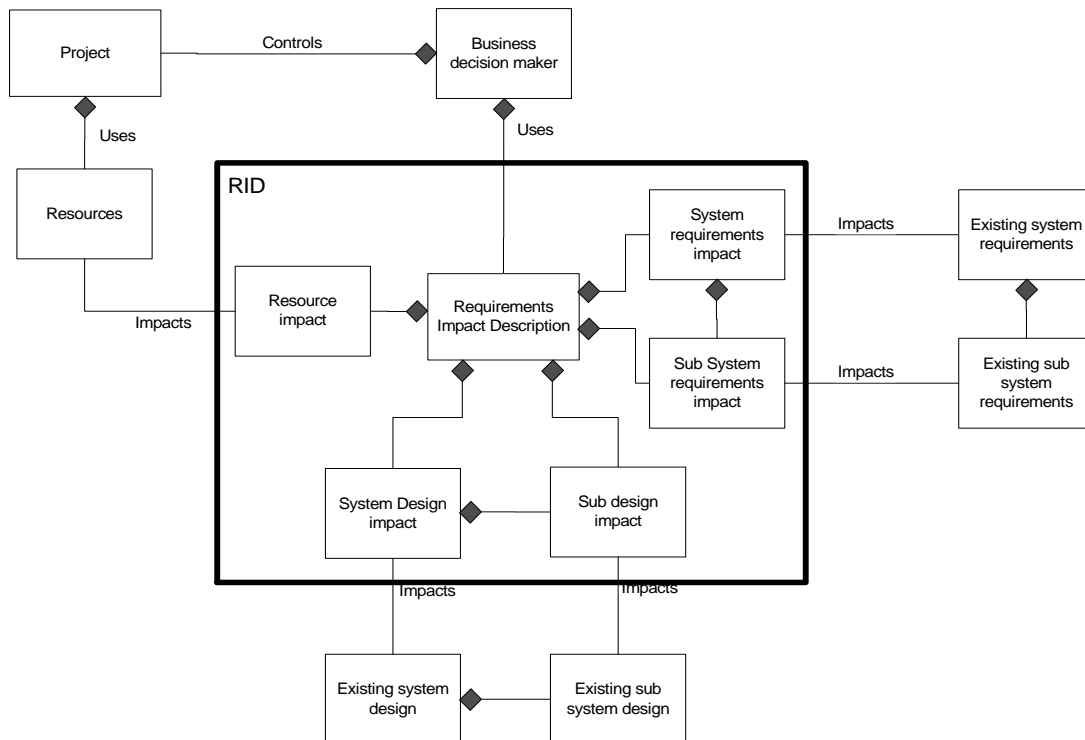


Figure 1: Overlapping Requirements.



The oval represents the entire requirements specification. The squares F1 - F3 represent the RID of features 1-3 and the impact they make. The areas O1 and O2 represent the overlapping impact on the requirements specification of respectively F1 and F2 and F2 and F3. When e.g. feature F1 is removed then the changes that are needed from feature 2 are still addressed by the RID from feature 2. Only in case both features are removed from the project the overlapping part also disappears as should be. This is similar for the impact on design since the RID also addresses design issues.

In UML notation this would like this:

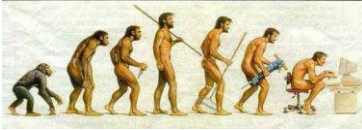


The RID is placed in the centre of this UML drawing. The RID consists of impact descriptions on system requirements and system design who at their turn consist of sub system requirements and sub system designs. The RID also describes the impact on those sub system requirements and designs. The RID also describes the impact (effort estimation) on the (human) resources of the project. The RID provides the information which the business decision maker needs.

4.2.5 Applicability

Use the Requirements Impact Pattern when,

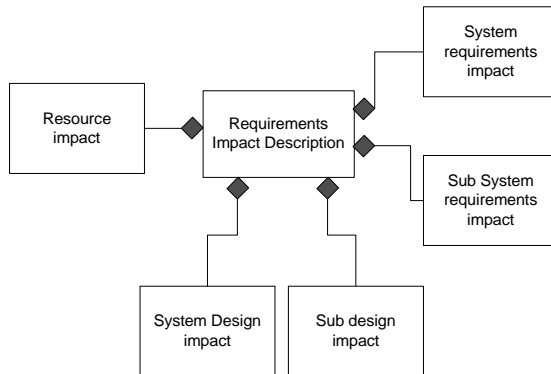
- You are in the early requirements and design phases
- You deal primarily with more or less isolated requirements. (Not for for-instance improving reliability or performance of an entire system).
- There is an existing system and system design with which you continue.



- Especially meaningful when the requirements list of the project is frequently changing during the early phases.

4.2.6 Structure

In UML (sub section of the picture above):

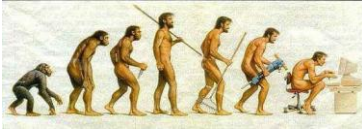


Feature title; Title	Project Name: Name	Effort estimate: # hours
<p>Author: name Date: Date Status: Status DocNR: DocNR</p> <p>Reviewers: Name 1, Name 2, Name 3 ... Name N.</p> <p>Authorizer: Name</p>		
<p>Requirements Specifications</p> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> <p><u>System</u></p> <p><u>Purpose</u></p> <p>Intention of feature</p> <p><u>General</u></p> <p>Brief descriptions of feature</p> <p><u>System aspects</u></p> <p>Impact on requirements of system aspects.</p> </div>	<p>Design Specifications</p> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> <p><u>System</u></p> <p><u>General</u></p> <ul style="list-style-type: none"> • General impact on system design. <p><u>System aspects</u></p> <ul style="list-style-type: none"> • Impact on design related to system aspects </div>	
<p>Impacted Subsystems</p>	<p>Impacted subsystems</p>	
<p>Impacted subsystem x</p> <p>Requirements Impact</p>	<p>Impacted subsystem x</p> <ul style="list-style-type: none"> • Sub system design impact 	
<p>Impacted subsystem y</p> <p>Requirements Impact</p>	<p>Impacted subsystem y</p> <ul style="list-style-type: none"> • Sub system design impact 	
<p>Impacted subsystem z</p> <p>Requirements Impact</p>	<p>Impacted subsystem z</p> <ul style="list-style-type: none"> • Sub system design impact 	
<p>Rejected Requirements</p> <p>Subsystem requirements which cannot be included in the projects.</p>	<p>Open issues</p> <ul style="list-style-type: none"> • List of open issues 	

4.2.7 Participants

The following people play a role when using the RID:

- The marketing department as representative of the customers plays a role in the discussions regarding the market wishes for the features. They have to know the



- value of the feature in the market in order to have a proper cost / benefit discussion.
- The system designer or architect writes the RID because he is the person that has enough overview to determine the impact of the feature on the requirements and design. He also has to have an idea about the amount of effort that is involved.
 - The project leader is ultimately responsible for the content of his project within the boundaries that are provided to him. The sub-project leader(s) are involved in providing more detailed data regarding the impact on resources on the parts of the project they are responsible for. They have discussions with the designers in order to get the first order estimates.

4.2.8 Collaborations

The input of the participants as mentioned above is gathered by the project manager. He collects all RID's for the project, does the math regarding the resources that are needed in total and whether it fits within the constraints of the project. When such is not the case the project manager starts the discussion regarding which features are out of the project and which are in. The wishes of the marketing department have to be balanced against the possibilities within development and against the (long term) benefits and commercial value of the features.

4.2.9 Consequences

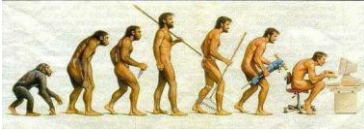
Since it is an iterative process we do not want to update the entire System Requirements Specification or Design specification. Every project however has to end with a complete set of documentation that is consistent within itself (the design has to match the requirements) and with the implementation. This means that after all discussions regarding the features that are to be delivered by the project the documentation has to be made up to date. This has some annoying impact on the later phases of the projects because then the effort has to be put in updating and finishing all the documentation. It needs no arguing that this is not a popular task and therefore holds a risk towards this project and especially its successors for whom the system documentation forms a base for further development. Update of the system documentation has to take place in order to be able to smoothly run the RID process of the next project. A list of RID cannot serve as a requirements specification document.

4.2.10 Implementation

The RID is applied in the early phases of development (phase 2 out of the 5 phases) and lose their meaning afterwards. This implies less effort regarding the configuration management of the sheets and therefore less overhead.

The input is the base lined system specification and design from the previous project(s). The RID serves as the 'delta specification'. Besides the RID also other forms of delta specifications exists such as Delta SRS and System overview. They serve a similar purpose but are merely in use in case of more correlated features and properties of the system that are implemented by the project.

The Project Team decides which variant(s) are used in the project, taking into account the stakeholder needs and the impact on the system.



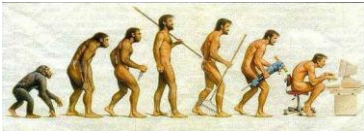
Creating sheets takes 2-5 days including discussions. A RID is a 1 sheet paper which makes editing easier.

Care should be taken to use isolated features of a manageable size i.e. in the order of 100 days to a few man-years. A project should contain a manageable amount of RID's i.e. up to a maximum of 20.

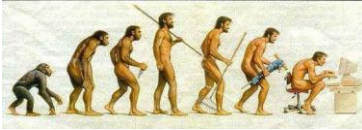
Care should also be taken that the feature does not grow in size itself by adding or removing requirements on the RID itself because this requires management of the RID's and of the versions of the RID's which makes it much more complex.

4.2.11 Sample

Find below a sample of a RID:



ExamCards SystemWide		Project: Spica	WBS: 300 days
Author: xxxx		Date:	Status: Concept DocNR: XJR -
Reviewers:			
Authorizer			
Requirements Specifications		Design Specifications	
System Purpose <ul style="list-style-type: none">Introduce viewing pages and overview protocolsExtend the ExamCards mechanism for automatic start of processing jobs and easy starting of processing viewing packages, all of it as defined in the EC protocol.Improve the ease of use of ExamCards General <ul style="list-style-type: none">Introduce viewing pages in all contexts in NSUI.Introduce the ExecutionList in all contexts in NSUI but only in the acquisition case of GV)Inclusion of NSUI-based processing and viewing steps in the Execution List<ul style="list-style-type: none">Automated start of processing for packages with user input, the processing is only invoked after user input (e.g. after pressing the submit button)Repeat processing from data-entries, or save in EC while the processing package is used.Processing protocols can have a help text in the info browserImport/Export of EC's, including processing & viewing steps.Define networking for all nodes (including DVD) using three level system, EC, series.ExamCard editor on the host, but also offline. The editor has a SW. Parallel usage of the same ExamCard is not supported.ExamCard can continue after a crash or system restart.Execution of ExamCards in batch mode for testing purposes (e.g. TEIMA) IQ Performance <ul style="list-style-type: none">Parallel start and execution of processing & scanning.Start of processing does not influence scanning. (<0.1 sec) Safety Option and Configuration <ul style="list-style-type: none">SW-key R2 Stierability <ul style="list-style-type: none">Upgrade of R1 EC'sBDAS & CDAS		System General <ul style="list-style-type: none">Use a queue mechanism for larger tasks, also in NSUI IQ Performance <ul style="list-style-type: none">PackageEngine: Memory usage < ??, CPU < ??ExecutionList per case: Memory usage < ??, CPU < ??Viewing Pages Memory usage / expected number of open packages??Extension of Execution Architecture model neededTwo way communication needed between processing jobs and EC to keep track the status. Safety Option and Configuration Stierability <ul style="list-style-type: none">Upgrade of EC's during installOnly add parameters to EC datamodelCDAS to BDAS downgrade of EC's needed for application specialists/Example Card creation.	
Magnet, Gradient, RF, Patient Support & Comfort, Physiology, Observation & Communication, ACQ Control		Magnet , Gradient, RF, Patient Support & Comfort, Physiology, Observation & Communication, ACQ Control	
Viewing & Processing <ul style="list-style-type: none">Processing protocols for some processing & viewing package which run in NSSearch tool in EC (to find protocols)Start package automated as define in the ExecutionListEC in GV & NSUI environments should be easily recognized as being the same thing.Multiple pages per context. Defined by review protocolsPrint EC contents		Viewing & Processing <ul style="list-style-type: none">Packages: ImageView, NeuroPerfusion, Diffusion, ImageAlgebra, PicturePlus & BreastPerfusion.	
Platform <ul style="list-style-type: none">The performed EC (including the steps to be performed) can be saved/retrieved from DVD/PACS/PatientDB.Status of an Exam/ExamCard must be known, also off 2WS or after restart.Protect EC contents (password protected)Backup/Restore EC database		Platform <ul style="list-style-type: none">The performed EC is present in the patient DB	
Patient Administration		Patient Administration <ul style="list-style-type: none">New parameters needed to keep the status of the ExamCard? Which?Blob needed to keep the complete EC.	
Rejected Requirements <ul style="list-style-type: none">Editing of package parameters in EQUI (only offline supported)Printing automated from ECPlanscan in NSUIDowngrading of ExamCards		Open issues <ul style="list-style-type: none">	



4.2.12 Known Uses

We know of only one usage of this specific sheet within Philips Medical Systems. The RID is in use now for several years and satisfies the needs.

4.2.13 Related Patterns

Feature Description in use by Alcatel.

4.3 Feature description pattern

4.3.1 Pattern Name and Classification

A **feature description** is a detailed description of a certain feature, or a number of related features, in terms of high-level system requirements.

4.3.2 Intent

The feature description translates the customer requirements in to more detailed high-level system requirements that are suitable to serve as input for the product development department. This document is also intended to unambiguously define what the feature is all about.

4.3.3 Also known as

The pattern is abbreviated as FD. No other names for this or similar patterns known.

4.3.4 Motivation

In many cases the requirements or wishes from the customer with respect to a product or solution are not suitable to be processed directly in the product development department because of one or more of the following reasons:

- The requirements are subject to different interpretations depending on the person that reads them.
- The technical part of the requirements is not enough elaborated.
- The requirements can not be placed in a broader overall business strategy.

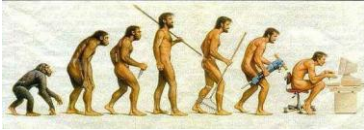
The feature description deals with all of the above mentioned shortcomings.

4.3.5 Applicability

The pattern is used before the start of the processing of a feature within the product development. It allows to make an initial objective assessment of the feature and to give an estimate of the needed effort and cost.

4.3.6 Structure

The feature description is a MS-Word document with has a number of mandatory explanatory chapters. Besides this, it also contains a table with the high-level system requirements, each uniquely numbered in order to be referenced further during the product life cycle.



4.3.7 Participants

Product Line Management together with the Product Architect produces the feature description document.

Product Line Management are people that are interfacing with the customers, collecting their requirements and place them in the product business strategy and roadmap.

Product Architects know the architecture of the system and provides the necessary technical details for formulating the requirements in an unambiguous way. They also make the initial assessment of the feature and provide the initial effort and cost estimates.

4.3.8 Collaborations

Product Line Management and the Product Architect sit together and discuss the customer requirement and how it affects the product. The result of this discussion is described in the Feature Description document, which is then reviewed by all involved parties.

4.3.9 Consequences

The goal of the pattern is to produce a stable high-level unambiguous requirements document that is the base document for the development of a customer feature. It can be considered as a contract between the Product Line Management organization and the Development Organization with respect to the feature that is requested by the customer.

4.3.10 Implementation

It is better to spend some limited time at the start of making customer requirements clear, rather than discovering later in the development cycle that we are making the wrong feature.

4.3.11 Sample

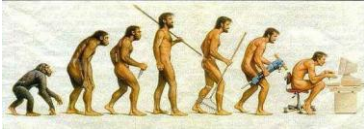
At the milestone where the requirement enters the product development and the development life cycle begins, the feature description must be available and reviewed.

4.3.12 Known uses

The pattern is applied within Alcatel.

4.3.13 Related patterns

Requirement impact description in use at Philips.



4.4 Light analyses of a Feature pattern

4.4.1 Pattern Name and Classification

Light Analysis for a feature.

4.4.2 Intent

Determine a budget within which we expect a quality solution for the feature can be developed, in a limited time frame (typical one to four days effort).

4.4.3 Also Known As

First Analysis / Initial Analysis / Initial assessment of a feature.

4.4.4 Motivation

If a customer requests a feature then we need to be able to determine the cost of the feature in a limited time frame since the effort we spend to define this cost should be as minimal as possible. At that point in time, we have not yet a commitment from the customer, so the effort spent could be lost in case it turns out that the customer is not interested any longer to invest in this feature based on the provided cost or for whatever other reason.

The term “customer” should be considered in a wide scope. It could be an external customer but also an internal customer in your company for instance the Product Line Management department can be the internal customer that requests the cost of a feature to the Research and Development department.

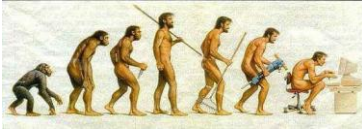
So the biggest challenge here is to balance the effort spent to perform the analysis against the accuracy of this analysis.

Following steps are taken in this process to come to an estimated budget:

- Understand the real customer problem to be solved
- Define the criteria that acceptable solutions will have to satisfy
 - Includes high level requirements
 - Includes quality attributes (e.g. performance, availability, security, modifiability, interoperability, ...)
 - Includes required effort to develop
- Enumerate possible solutions
- Evaluate and rank the solutions
- Derive a budget estimate with which it should be possible to build an acceptable solution
- Do a first risk analysis

4.4.5 Applicability

This pattern can fit in all development processes.



4.4.6 Structure

The result of the pattern consists of a Word document or a PowerPoint presentation that can be used to present the outcome of the Light Analysis.

4.4.7 Participants

- Product Line Management are people that are interfacing with the customers, collecting their requirements and place them in the product business strategy and roadmap.
- Product Architects know the architecture of the system. They make the initial assessment of the feature and provide the initial budget estimates.
- Domain Architects know in depth the architecture of parts of the system.

4.4.8 Collaborations

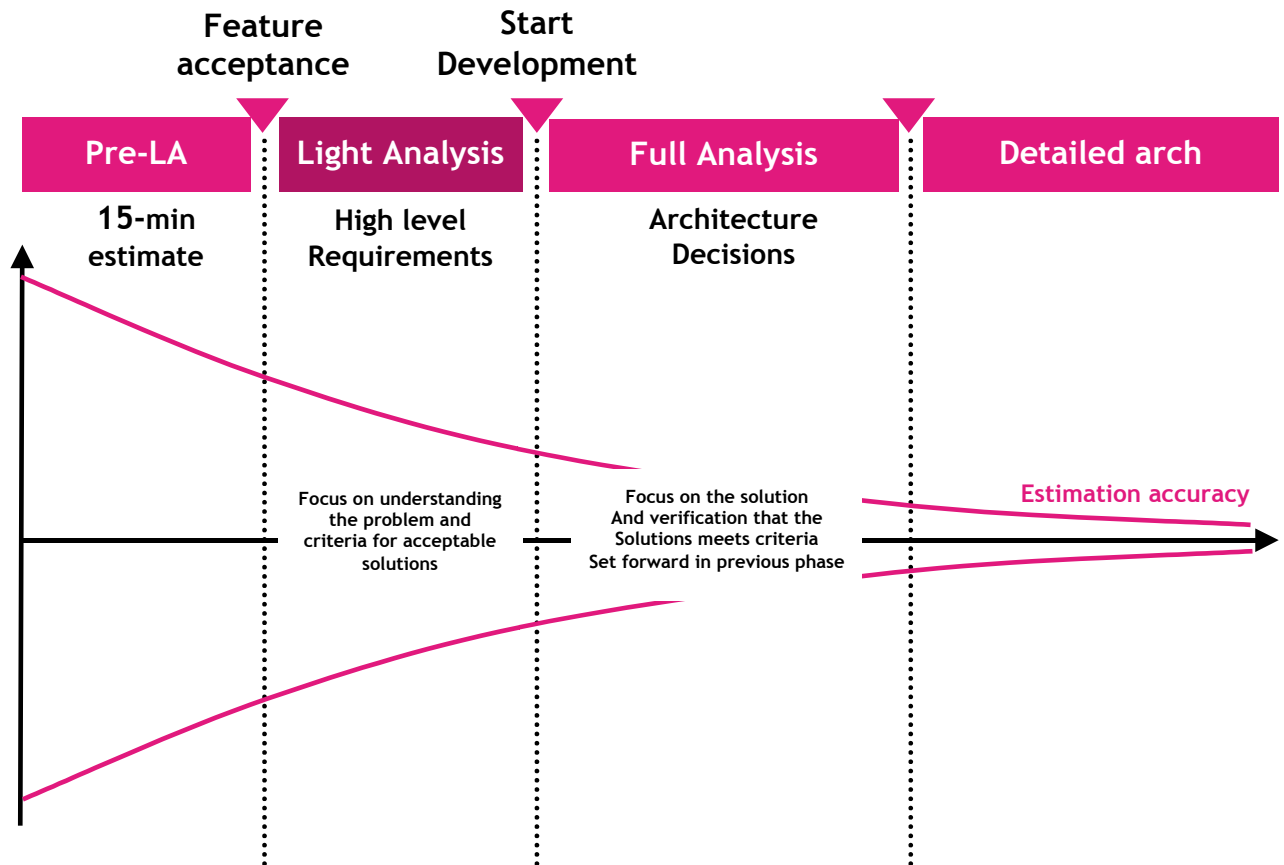
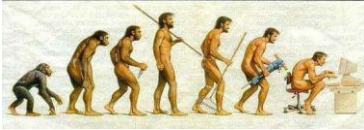
Product Line Management collects the requirements from the external customer and translates these into a feature request. The Product Architect performs the Light Analysis and consults Domain Architects in different area's to get more details as needed.

4.4.9 Implementation

Budgets estimates at this stage of the development cycle are not very accurate. One of the pitfalls when applying this pattern is that these budget estimates are in most cases too optimistic. This is because there are usually unforeseen issues that only come up later in the development cycle when more details become available. If this happens then one loses the possibility that when taking a few features together the budget inaccuracies compensate for each other. So more realistic budget estimates should be done, taking into account that unforeseen issues might pop up later.

4.4.10 Sample

The pattern can be applied as illustrated in the picture below.



4.4.11 Known Uses

The pattern is applied in Alcatel-Lucent.

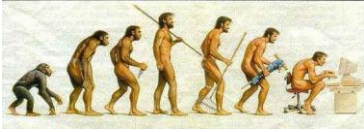
4.4.12 Related Patterns

The Feature Description pattern is related to this one. The intention of the Feature Description pattern is to translate customer requirements into high-level technical requirements. The intention of the Light Analysis pattern is to determine a budget to develop the customer requirement. In most cases, both patterns are applied by the same people.

4.5 Delta Specifications pattern

4.5.1 Pattern Name and classification

Delta Specifications (DELTA'S) for writing incremental requirements and design documentation.



4.5.2 Intent:

To be used to define incremental requirements for evolutionary product development starting from the specifications of an existing product. The Delta's describe a high level requirement in an early stage of development focusing on its impact on existing requirements and design.

The delta specifications (DELTA'S) are covered in three types of documents:

1. Delta SRS (Delta System Requirement Specification): focus on new/modified system requirements
2. SO (System Overview): focus on technology (design) updates
3. PRES (Project Requirements Sheet): focus on adding isolated features (described as separate pattern in paragraph 4.2)

The project team selects what type(s) of documents to use based on the project content.

4.5.3 Motivation

In an evolutionary development approach, a large number of components will be re-used from the previous product and a limited number of components will have to be changed. A project team working with delta requirements wants to focus on the management of the changes. These changes are described in above documents relative to the previous product specifications. The previous product is described in the System Requirement Specification (SRS) & System Design Specification (SDS).

Writing and updating the SRS & SDS for an entire system is often a lot of work when the system is big and complex. Every update big or small of a System Requirements Specification brings overhead with it for reasons of configuration management. When changes are frequent the amount of overhead becomes un-workable. The delta specifications speed up the initial phases of the project because it supports the discussion and decision making process regarding the contents of the projects.

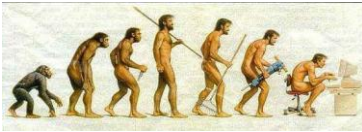
4.5.4 Applicability

Use the Delta Specifications for system developments or sub system developments as part of the product creation process when:

- You are in the early requirements and design phases.
- There is an existing system and system design with which you continue.

4.5.5 Structure

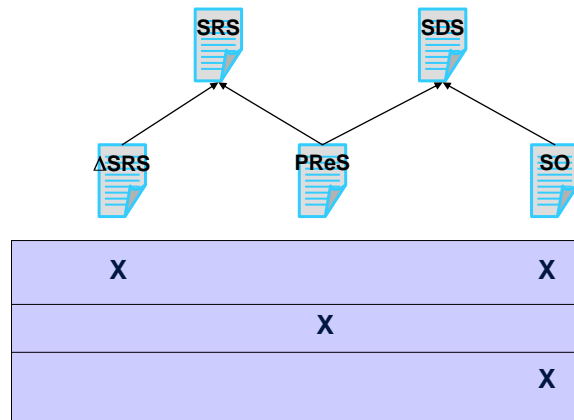
The following drawing gives a graphical presentation of the relation between the product specification documents:



Δ SRS, PReS, SO: how they relate

Delta spec's

- new / modified system requirements
- adding system features
- technology update
little new requirements



At the project start: Use as baseline the previous product documentation, i.e. SRS & SDS.

At start of detailed design: The delta specifications together with the baseline system documentation specify the new product.

Before start of testing: The delta specifications are merged into the SRS & SDS and lower specifications and are of no use anymore.

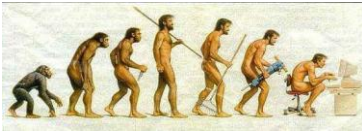
4.5.6 Participants and collaborations

The following people play a role when using the delta specifications:

- The marketing department as representative of the customers plays a role in the discussions regarding the market wishes.
- The system designer or architect writes the Delta Specifications because he is the person that has enough overview to determine the impact of the feature on the requirements and design.
- Project Manager and sub-project managers use the delta specification for input to effort estimates.

4.5.7 Consequences

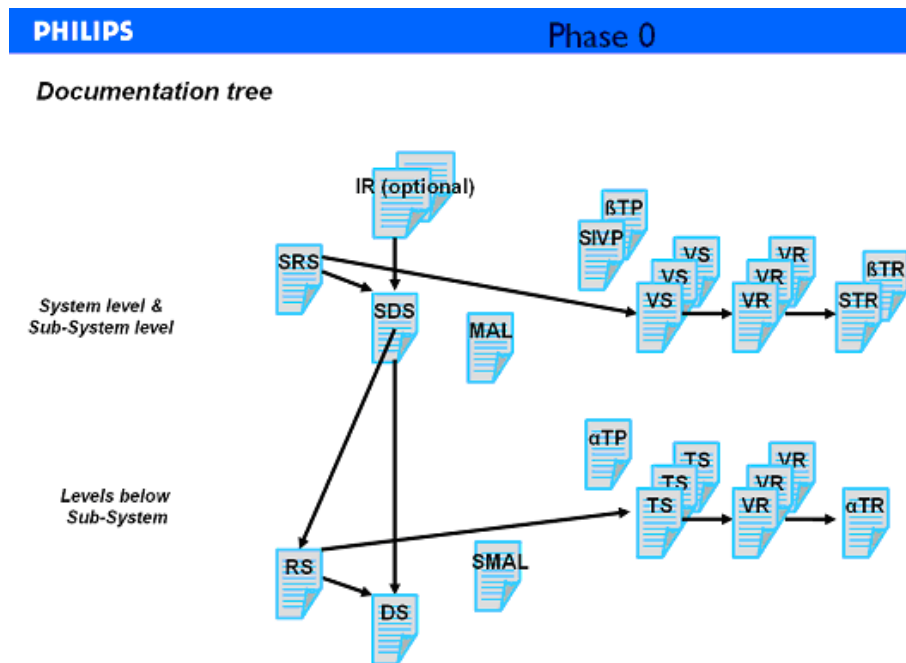
Since requirements gathering is an iterative process, it is helped by focusing on the changes in evolutionary developments. However, before the testing phase the delta requirements have to be merged into the System Requirements Specification and/or System Design specification. Every project has to end with a complete set of documentation that can be used for the next project as basis for further development.



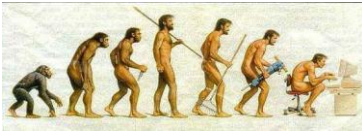
In case several projects run simultaneously, while making use of shared components (e.g. software), these projects will have to work together in order to merge their contributions into one shared SRS & SDS.

4.5.8 Implementation

The input is the base lined system specification and design from the previous project that made the existing product (Phase 0).



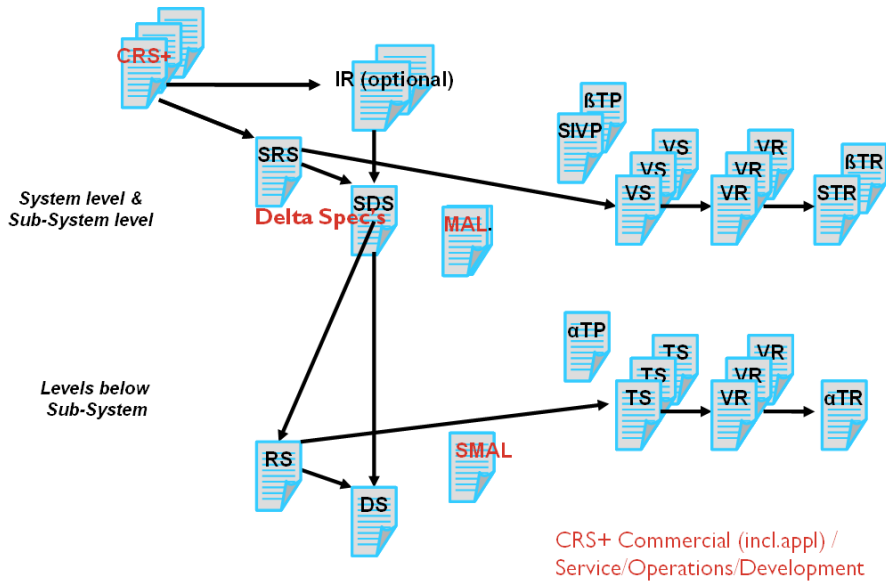
In the early project phases (feasibility and global design) the delta specifications are gathered. Together with the baseline specifications the new product is specified (Phase 2/3 review).



PHILIPS

Documentation tree

Phase 2/3 review

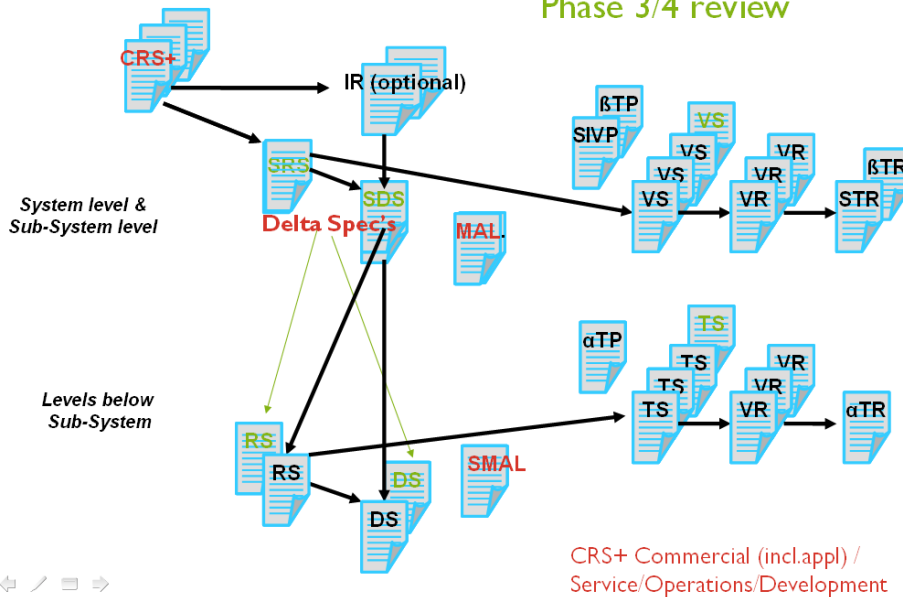


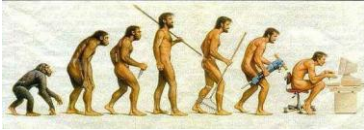
In the detailed design phase the low level requirements and designs are updated based on the delta specifications. At system level the SRS and SDS are updated before the test phase starts (Phase $\frac{3}{4}$ review).

PHILIPS

Documentation tree

Phase 3/4 review

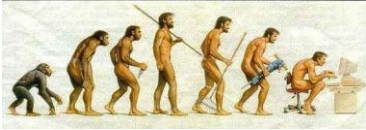




4.5.9 Samples

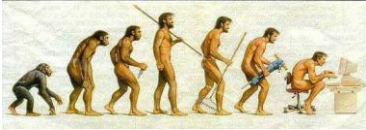
Examples of delta SRS and System Overview are given below

Delta SRS example		
Purpose		
The requirements in this section are meant for project scoping and serve as design direction. They are detailed in the specifications in the other sections, and in sub-system requirement specifications.		
RID	Requirement	Supports RID
M.SY.G.1	Requirement G1	CRS
M.SY.G.2	Requirement G2	CRS
General: Functional Requirements		
RID	Requirement	Supports RID
M.SY.F.1	Requirement F1	M.SY.G.1
Design / Appearance		
RID	Requirement	Supports RID
M.SY.F.1	Requirement F1	M.SY.G.1
Image Quality (FOV/SNR/SNR uniformity)		
Performance		
Safety and Standards		
Option and Configuration		
Service, Testing, Development, Manufacturing and Reliability		
Rejected Requirements		
Obsolete Functionality		



System Overview example

System Requirements Specification	System Design Specification	WBS / remarks
<u>Purpose</u> <u>General</u> <ul style="list-style-type: none"> • <u>IQ</u> <ul style="list-style-type: none"> • <u>Performance</u> <ul style="list-style-type: none"> • <u>Safety</u> <ul style="list-style-type: none"> • <u>Option and Configuration</u> <ul style="list-style-type: none"> • <u>Service, Testing, Manufacturing and Reliability</u> <ul style="list-style-type: none"> • 	<u>General</u> <ul style="list-style-type: none"> • <u>IQ</u> <ul style="list-style-type: none"> • <u>Performance</u> <ul style="list-style-type: none"> • <u>Safety</u> <ul style="list-style-type: none"> • <u>Option and Configuration</u> <ul style="list-style-type: none"> • <u>Service, Testing, Manufacturing and Reliability</u> <ul style="list-style-type: none"> • 	<p>1</p>



System Blocks Overview

Insert drawing of (new, modified, removed) (Technical) Building Blocks, taken from, and relative to, the last authorized SDS

Background information

System function and design rationale. Fill in where appropriate.

Open issues

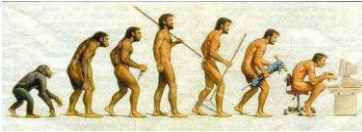
Rejected requirements

1 Building Block X

BB X Requirements Specification	BB x Design Specification	WBS / remarks
•	•	

Background information

Open Issues.



4.5.10 Known Uses

Philips Medical Systems

4.5.11 Related Patterns

The Requirement Impact Description (RID), also called “Pres”.

4.6 Manageable requirements traceability

4.6.1 Intent

When developing large and complex products it is often difficult to check if the end-product contains all required functionality. To work around this, Requirements Traceability is developed. Requirements traceability allows each specific requirement to be traced from product level down to implementation level.

This way of tracing requirements has some known drawbacks, as described below. For this reason a “simplified” way of requirements traceability is introduced.

4.6.2 Also known as

No other names known.

4.6.3 Motivation

The biggest issues with standard requirements traceability are:

- The number of requirements to be traced tends to explode into an unmanageable number.
- The relation between lower level requirements and product requirements becomes unclear.

This results in the following problems:

- It is hard to determine if all requirements are designed, implemented and tested completely.
- The impact of requirement changes in the product is unclear.

This pattern describes a way to avoid these problems, thus to keep requirements traceability ‘manageable’.

4.6.4 Applicability

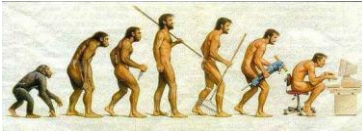
To keep the number of requirements manageable, two important rules are introduced:

Rule #1: Product requirements are introduced on product-level, and comprise the:

- External interfaces of the product.
- Behavior and functionality of the entire product.
- The product’s non-functional requirements.

Not part of the requirements:

- Product internal decomposition.
- Product internal interfaces.



These are considered design of the product, and are a result of specific product requirements.

Traceability is done until the first discipline specific requirements documents.

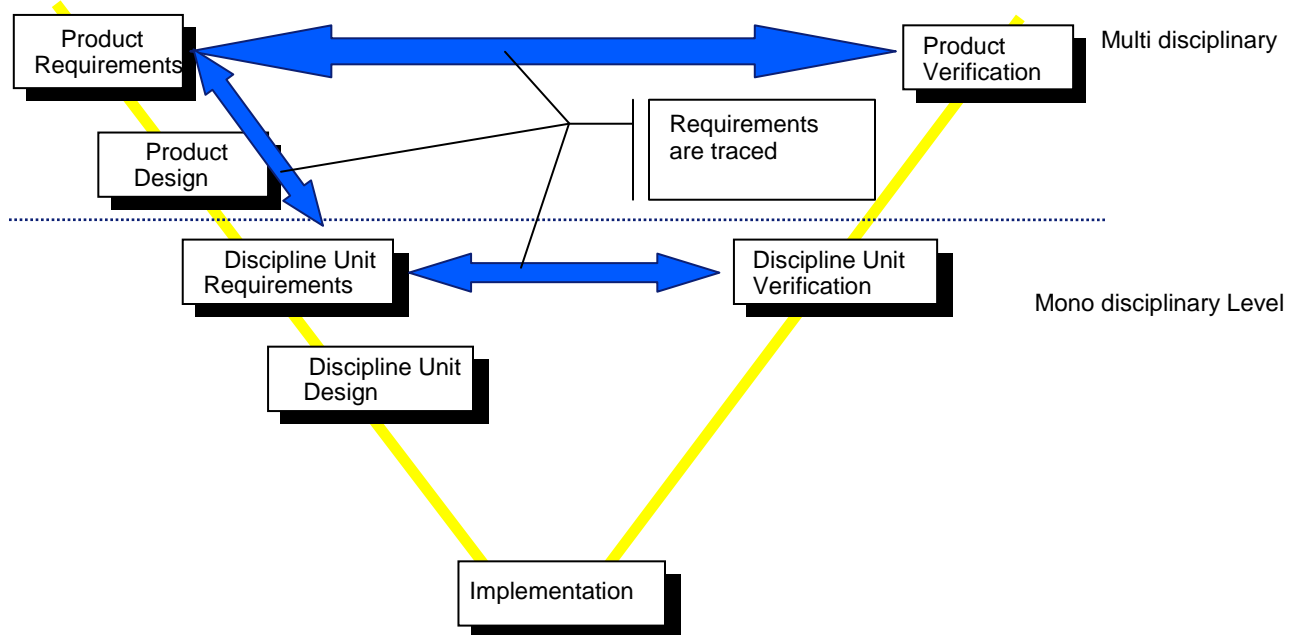
Rule #2: No new requirement tags are introduced in lower level documents with respect to traceability.

4.6.5 Structure

In this document, as an example, a virtual product is described, consisting of one unit, which contains some electronics, mechanical parts and software (from here on called “discipline”). This does not imply that a product has to be limited to this architecture.

It is important to remember that in the hierarchy on Product level multi-disciplinary units are used and on unit level, the unit is split into three mono disciplinary units, each describing their part of the unit.

Product traceability structure is as follows:



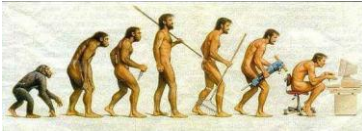
Product requirements are determined by the System Designer and documented in the Product Requirements Specification. This is the only source of product requirements.

The *Product Design Specification* decomposes the product into one or more mono- disciplinary units.

The *Discipline Unit Requirements and Design* documents are created for each discipline and describe the requirements and design of the unit designed in the Product Design.

Requirements are allocated to the units and disciplines using the Requirements Traceability Matrix (not in the picture).

The *product verification* includes a test traceability matrix that shows for each product requirement how it's tested, and the test result (not in the picture).



The *Discipline Unit Verification* specification includes a list of all requirements allocated to that discipline and how each requirement is tested.

4.6.6 Participants & collaborations

- System Engineer: Defines the Product Requirements in the Product Requirement Specification, Product Design Specification and Requirements Traceability Matrix.
- Software / Hardware / Mechanics designer: References the allocated requirements in the discipline Unit Requirement Specification.
- Product Verification Engineer: Creates the product verification document and product test traceability matrix.
- Software / Hardware / Mechanics verification engineer: Creates the discipline verification specification and allocation of requirements to test cases.

4.6.7 Consequences

- Such a system of maintaining requirements works well, if it is kept consistent at all times. Changes in requirements must be communicated to all parties involved (Requirements Traceability Matrix)
- This approach has the advantage that when a requirement changes, using the requirements traceability matrix the effect of this change can easily be determined.
- Not only requirements need to be traced, but also documents. If the Requirements Traceability Matrix allocates a specific requirement to a specific discipline unit, it is crucial that the Discipline Unit Requirement specification can be found easily.
-

4.6.8 Implementation

A standard requirement naming convention is used:

PRS.RequirementClass.Requirement

Where:

PRS: The place where the requirement originates. This is always one of the top-level requirements documents.

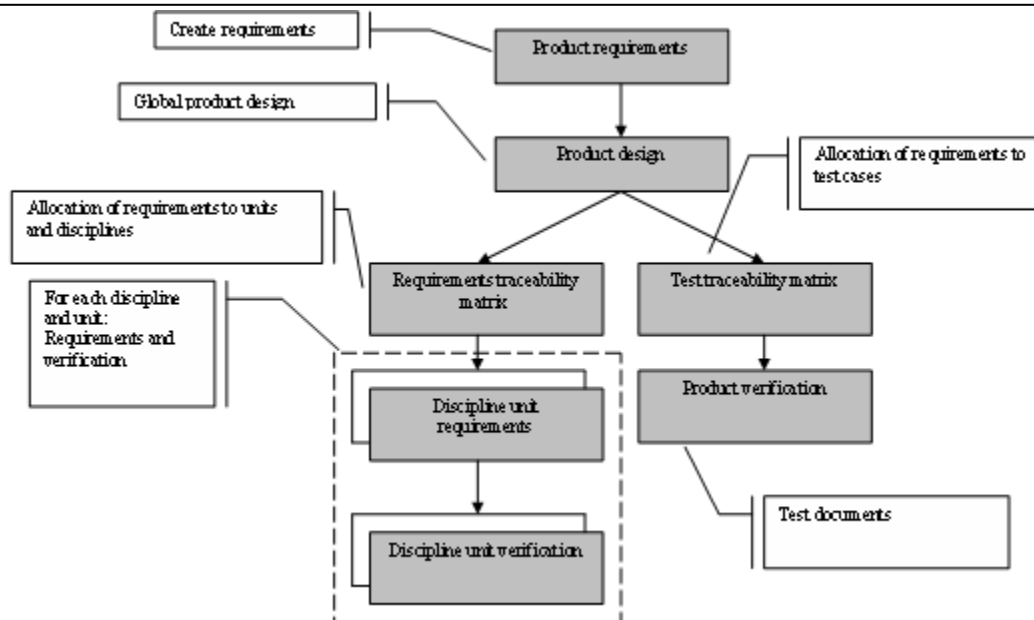
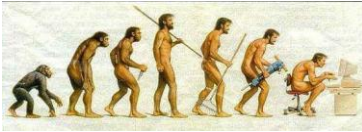
RequirementClass: The type of requirement, for example: Manufacturability, or Reliability.

Requirement: Short requirement description.

Using such a strict requirement naming convention, including the place where the requirement comes from, allows tracing of one specific requirement through all layers of documentation both from product level downwards to the discipline specific requirements and reverse.

Also, this naming convention allows some kind of “subclassing” of requirements (for example: adding “.MySubclassedReq” to a requirement) without losing traceability.

The document hierarchy looks globally like this:



Product requirements	Defines product requirements
Product design	Globally decomposes the product into one or more multidisciplinary units.
Requirements traceability matrix	Allocates product requirements to units / disciplines
Test traceability matrix	Allocates product requirements to test cases on product level.
Product Verification	Specification of test cases and results
Discipline Unit Requirements	Discipline requirements specification, more detailed description of requirements for a discipline.
Discipline Unit Verification	Verification specification and results.

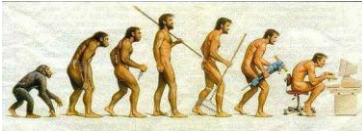
4.6.9 Sample

Below is an overview on how bidirectional requirements traceability is implemented:

Product Requirements Specification:

PRS.Performance.StartupTime
➤ The product shall start within xx seconds.

In the Product Design Specification, each chapter is preceded by a list of requirements applicable to the specific chapter:



Timing

PRS.Performance.StartupTime

Total start-up time is divided over:

Unit 1: yy seconds

Unit 2: zz seconds

The Requirements traceability matrix allocates the various requirements to units / disciplines:

Requirements Tag Identification	Unit1 Software	Unit1 Hardware	Unit1 Mechanics	
PRS Requirement Tags				
PRS.Performance.StartupTime	X	X		

Requirements allocated to each discipline in each unit

The Unit Requirements Specification of each unit contains all requirements allocated in the Requirements traceability matrix.

PRS.Performance.StartupTime

Unit will respond within yy seconds.

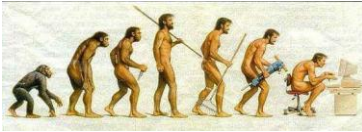
Some requirements do not require any more explanation; these are collected in a single chapter in the Unit Requirements specification, like this:

Requirements applicability

When a requirement is clearly documented in higher level documents, it is listed below. Requirements that need any more clarification or specialization are discussed in the remainder of this document.

PRS.Performance.StartupTime

The verification documentation of each discipline contains a list of requirements, and in which test case these are tested:



Requirement ID	Test case	Description
PRS.ExposureUpgrade.Compatibility	Application functionality (White box)	The version numbers of Buildingblocks will be displayed
PRS.ExposureUpgrade.New&Obsolete	Application functionality (White box)	Test item: Applitv test
PRS.ExposureUpgrade.OperationalConcepts	Application functionality (Black box)	Test acquisition techniques

The product Verification documentation contains a Test traceability matrix that allocates product requirements to test items:

Requirements to be tested	Test case					Features to be tested (and indicated progress or result)														Test Result					
	Test	Basic Test	Alternative Test	Use Case Test	Alternative Test	Tested by re_use	Tested by design	Tested by systemgroup	Functionality	Performance / Timing	Responsiveness	Usability / Customizing	Image Quality	Installability	Maintainability	Safety	Reliability	Manufactureability	Approbation	Security	Documentation	Increment 1	Increment 2	Product Test	Release Test
PRS.ExposureUpgrade.Compatibility						x	x						x	x					x		x				Passed

List of product
requirement
tags

Test case in
which this
requirement
is tested

Explicit /
implicitly
tested and by
who??

Which
aspects have
to be tested?

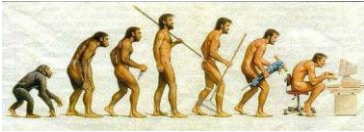
To make checking the consistency more easy, the Product Requirements specification, Product Design specification and Unit Requirement Specification contain a list of all requirements referred to:

Appendix A: Requirement Tags Look-up table

PRS.ExposureUpgrade.New&Obsolete	6
PRS.ExposureUpgrade.OperationalConcepts	6
PRS.ExposureUpgrade.Compatibility	7

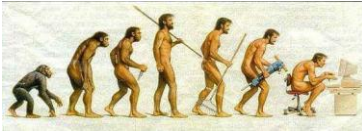
4.6.10 Known uses

Philips Medical Systems



4.6.11 Related Patterns

No related patterns.



5 Design related Process Patterns

5.1 Critical Computer Resource Management pattern

5.1.1 Pattern Name and Classification

Resources like CPU memory, CPU capacity, disk space, etc. are limited by nature. These resources are known as “**Critical Computer Resources**” or CCR. During the evolution of a product and its SW, these resources should be managed carefully and with special focus in order to preserve as much as possible future evolutions of the product. This is more then ever applicable in case of embedded SW and HW design where these resources are scarcer.

5.1.2 Intent

When adding new features or HW to an existing product, the CCR should be properly estimated and tracked as an explicit item during the complete development cycle. It avoids unpleasant and usually very difficult and costly to fix “surprises” late in the development cycle. Besides this it also avoids premature dead of a product variant because of lacking resources to accommodate new features

5.1.3 Also known as

Critical Computing Resources.

5.1.4 Motivation

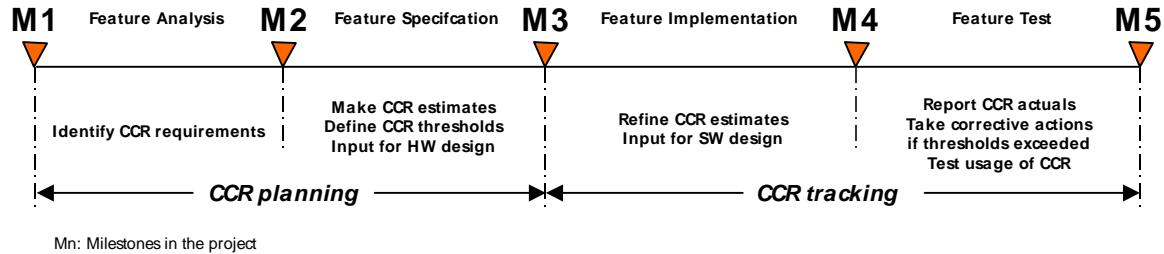
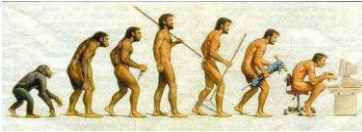
Engineers who are developing features, especially in SW, intend to forget about CPU memory and CPU performance limitations. This is even more the case when using object oriented languages since usually a lot more code is generated by the compiler and memory is used by this generated code compared to traditional lower level programming languages. Explicitly paying a lot of attention to these critical computer resources during the complete development lifecycle avoids in most cases that issues related to this topic only popup very late in the cycle (during test). When these issues pop up late in the development cycle, there are usually not a lot of options open to correct the issue without having to start the design (partly) all over again. This is not only very costly, but also introduces severe delay in the project.

5.1.5 Applicability

The process pattern is applicable for all SW design. It is also applicable for HW design in case programmable components (Field Programmable Gate Arrays) or processors (On Board Controllers, Network Processors) are being used.

5.1.6 Structure

Below is an overview of applying the process when dealing with the feature development process. The same principles can be applied for other development processes aswell.



The following phases can be distinguished:

- During feature analysis, CCR requirements are identified and formulated. For example: number of subscriber lines to be supported, sustainable packet processing speed, etc.
- During feature specification, based on the available CCR requirements, CCR estimates are made in terms of the amount of memory required, needed CPU capacity, disk space needed. For new HW design, in addition some free margin must be taken into account in order to allow the product to evolve in time if needed and to be able to cope with additional features on the same HW platform. For example, the amount of free memory in case of new board design should be at least 50% of the available memory. In addition, some CCR thresholds are being defined. These thresholds specify the boundaries in which the realized CCR should fall compared to the estimated CCR. When these boundaries are crossed, the CCR issue is brought under the attention of the project meeting, which can decide on corrective actions. These thresholds allow for some flexibility and inaccuracy of the estimated CCR during the development process.
- During feature implementation, the CCR estimates done during feature specification are taken as requirement (CCR budget) for the feature and are used to steer the SW design.
- During the different test phases of the feature, the actual CCR usage by the system is measured and compared against the CCR requirements and estimates that were put forward. Corrective actions and lessons learned are taken as necessary.

5.1.7 Participants

Product Line Management together with Product Architects formulate the CCR requirements according to the customer requirements and the product future needs.

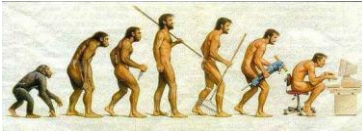
The Feature Architect translates the CCR requirements in CCR estimates based on the solution selected for the feature. The Product Architect reviews these CCR estimates in order to keep a System wide view and control on the overall CCR of the product.

The SW and HW designers take the CCR estimates into account for making their design. Finally they measure the CCR actually used by the feature and try to stay within the CCR budget that was put forward. From the moment they cannot maintain any longer their CCR usage within the acceptable boundaries, they have to report this as soon as possible to the project management board where the necessary corrective actions can be decided.

During this whole process, the CCR are maintained in a sheet per product or product part depending on how it is organized for the product concerned.

5.1.8 Collaborations

When applying the process pattern, there is close collaboration and discussion between the different roles contributing to the CCR from the start of the project all the way to the end of the project as has been explained in previous chapter.



5.1.9 Consequences

By formalizing the CCR as special item to be estimated and tracked during the whole development cycle, one can in most cases avoid late surprises or premature dead of a product. It is also more cost effective, since future evolutions of the product shall be considered when doing the analysis, in order to come to the most economical solution in terms of the HW required (memory, CPU processing speed, etc.) and the cost associated, to allow sufficient evolution of the product being able to meet the future requirements.

5.1.10 Implementation

Since estimating CCR is not always trivial (for instance in case of estimating needed CPU processing capacity that fits the requirements), applying the process smoothly requires some experience which can be build up step by step doing the process pattern project after project. So initially still some issues related to CCR could pop up late in the cycle. Estimates will not be always be accurate from the beginning. However the experiences gained during the execution of the process, followed by appropriate lessons learned analysis, will make it possible for the participants to gain more and more knowledge in this complex area and gradually will make applying the process pattern more mature in future projects.

5.1.11 Sample

Below is an example of how a CCR estimation and tracking sheet could look like.

CCR Estimation/Tracking Worksheet Release Rx.x Baseline Release Rx.x-1

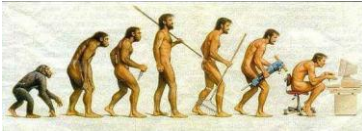
Board	CCR Type	Physical Limit	Current Estimated Amount Free		Actual Amount Free (date)		Amount free Threshold (date)		Amount free Baseline Rx.x-1
			Value	(%)	Value	Diff btw Actual & Estim	Value	(%)	
Card1	RAM	33.554.432	792.128	2,36%	794.524	0,30%	8.388.608	25,00%	822.364
Card2	RAM	33.554.432	10.463.102	31,18%	10.873.648	3,92%	8.388.608	25,00%	11.542.964
Card3	RAM	33.554.432	15.868.426	47,29%	12.345.678	-22,20%	8.388.608	25,00%	17.568.972
Card4	RAM	134.217.728	60.648.242	45,19%	58.462.482	-3,60%	33.554.432	25,00%	62.462.868

Corrective Actions

Board	CCR Type	Corrective Action
Card1	RAM	No further feature evolution on this board

When a new project starts a new sheet is created with all the cards available in the product and with the actual CCR (in this case RAM memory) from the baseline project release. During feature specification, the new estimated CCR values are written down in the appropriate column.

Later on when actual CCR become available, the actual amounts of CCR are filled in. In case certain thresholds are passed, then the percentages are displayed in red, meaning that most probably corrective action needs to be taken for the board concerned.



5.1.12 Known uses

Estimating Critical Computer Resources as part of estimating the cost and effort of a project is also mentioned in the Capability Maturity Model (CMM) and Capability Maturity Model Integration (CMMI). These models, well known by the industrial community, are used a lot by companies trying to get their development process streamlined and under control. So most probably, other companies might also have implemented similar process patterns to deal with these CCR.

5.1.13 Related patterns

Similar patterns will most probably exist trying to address the same problem, but are not known by the author.

5.2 Multidisciplinary product configuration management pattern

5.2.1 Pattern name and classification

The **Multidisciplinary product configuration management** pattern helps to design a product taking into considerations the viewpoints from various stakeholders.

5.2.2 Intent

The intent of the process pattern is to make the organization more aware of the importance of multidisciplinary product configuration management in an early phase of product development. In order to realize this, a common understanding on various views of a product needs to be established.

The result of this process pattern is a tutorial/guideline for usage of multiple product representations and related terminology,

5.2.3 Also known as

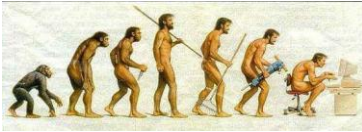
-

5.2.4 Motivation

Marketing, Product Designers, Product Configuration Managers, Development Engineers (of various disciplines), Manufacturing Engineers and Service engineers each use their own product structuring and talk about it from their own perspective and with their own terminology.

Traditionally each discipline tends to work as long as possible in their familiar mono-disciplinary authoring environment e.g. CAD-M, CAD-E, and Software Development Environment.

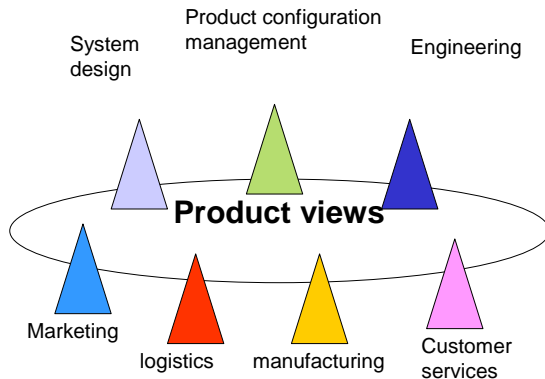
All disciplines need to be aware of these multiple view points, terminology and product structures and start working in a multidisciplinary configuration management environment early in the design phase. This allows us to make the proper architecture choices (i.e. modularity, interfaces) when they have the least impact on development resources. This will improve design reuse, manufacturability, testability, serviceability, simplify order management etc.



5.2.5 Applicability

The pattern is targeted and therefore applicable for all disciplines with product development.

5.2.6 Structure



Each disciplines looks at a product from their own perspective, and therefore has its own requirements and wishes with respect to the representation of a product. In order to optimally support these needs, currently several product representations (also referred to as product structures) are in use. Obviously there is a strong relation between these representations.

Each product representation consists of:

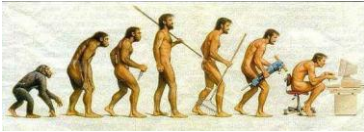
- Objects (i.e. parts, documents)
- Object description data
- Relations/links between objects (i.e. structures)
- Attributes on objects & relations (to allow views on structures)
- Status of objects & relations
- History of objects & relations

The following product representations are recognized and described:

1. Conceptual Product Structure
2. Commercial Product Structure
3. Engineering Product Structure
4. Manufacturing Product Structure
5. Customer Support Product Structure

The Conceptual Product Structure describes the product from a design perspective. The main objects in this representation are called conceptual elements, which represent the design decomposition. Each Conceptual Element must follow certain characteristics:

- The conceptual elements reflect a stable product architecture and these elements have there own design life cycle
- The design of the conceptual elements can be independently verified & validated
- The conceptual elements have clearly defined/described interfaces.



- The conceptual elements have their own Design History File i.e. documentation set including the following information:
 - Design input i.e. requirements
 - Design output i.e. specifications & descriptions
 - Design review
 - Design verification & validation
 - Design transfer
 - Design changes

Within this representation the following terminology is defined:

- Family, system, system variant, subsystem, conceptual element & platform

The Commercial Product Structure represents the product from a sales perspective.

It describes how, based on selection of features and options, a specific variant of a configurable sales product can be selected. This structure is used in the product catalog from which an end-customer orders a product.

Within this representation the following terminology is defined:

- configurable sales product, feature, option, commercial constraints

The Engineering Product Structure is the multidisciplinary Bill-of-Materials described from an engineering perspective. It contains the materials and related technical product documentation. Documents (e.g. CAD-drawings) and data files (e.g. SW executables) are imported for the various authoring tools i.e. CAD-systems, Software Development Environment. The material contains the basic engineering master data.

Within this representation the following terminology is defined:

- Materials , Approved Manufacturer List

In our IT solution we have separated the engineering product structure into a configurable (top-level) Bill-of-Material and fixed (lower-level) Bill-of-Materials.

This configurable top-level part is referred to as the Product Variant Structure (PVS).

The PVS describes the conditional relations between the fixed BoM's in terms of dependencies and characteristics.

Within this PVS the following terminology is defined:

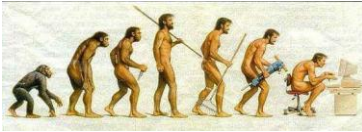
- Configurable product, product variant, dependency, characteristic

The Manufacturing Product Structure is the classic multidisciplinary Bill-of-Material. Currently the materials and relations in the manufacturing product structure are the same as for the Engineering Product Structure. Only the master data is enhanced with information required within manufacturing, procurement, logistics, packaging, distribution etc.

Within this representation the following terminology is defined:

- Materials, Building Blocks and assemblies

Building Blocks and assemblies should be seen as classification (labeling) of materials in the manufacturing Bill-of-Material. A material that is labeled as Building Block needs to have clearly defined functionality and interfaces and are fully testable. An assembly is an intermediate state in the supply chain or manufacturing, created for supply chain / manufacturing purposes only.



The Customer Support Product Structure represents the product configuration from a customer support perspective. This reflects the physical hierarchy of the product how it is serviced in the hospital. This structure is used in e.g. service documentation, reporting of maintenance activities and traceability of delivered devices.

Within this representation the following terminology is defined:

- System, Physical Main Block, Physical Sub Block, Traceable Item

5.2.7 Participants & collaborations

This pattern focuses on a guideline to be used in all disciplines.

The guideline is developed by the PLM architect in close cooperation with the process owners of Software Configuration Management, Hardware Configuration Management, Manufacturing Engineering and Marketing.

5.2.8 Consequences

There is a strong relation between the various product representations. Each product representation reflects a process behind it. Therefore a good understanding of all product representations is a precondition to understand the consequences of design choices in product structuring. This knowledge is a precondition for proper modular design and incremental product development and ultimately reducing design complexity and increasing speed in development.

5.2.9 Implementation

Aspects to be taken into account for this pattern are:

- Add real-life examples that will be recognized by the target group (trainees)
- Realize that old terminology will follow you for years. Consequent usage of the terminology by all process owners is necessary. Do not assume, but verify that people use the terminology as intended.
- Make the training as visual as possible so that it will stick in the mind of people.
- PowerPoint's & posters are helpful in communication but significant training is required before it results into changes in our way of working (see 9: consequences) An additional process pattern will be assigned to this.

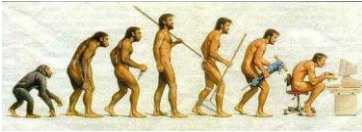
5.2.10 Sample

A detailed example of these different representations for one specific product is not yet available. The usage of the Conceptual Product Structure and the Product Variant Structure is currently being piloted. The design documentation which is now planned to be structured conform the conceptual elements was until now project oriented. Ideas on the usage of the Product Variant Structure are still in a premature stage.

5.3 Software FMEA pattern

5.3.1 Pattern name and classification

Software failure mode and effect analysis



5.3.2 Intent

Failure mode and effect analysis (FMEA for short) is a method to think about failure modes in a system and the effect they have. FMEA enables thinking about failure modes in an early development phase. FMEA has been around for a long time but has never been documented for the purpose of improving product quality when it comes to software.

5.3.3 Also known as

FMECA (Failure modes, effects and criticality analysis) src: www.ntnu.no/ross/srt/slides/fmeca.pdf

5.3.4 Motivation

A FMEA session is being applied to increase product quality and reliability.

Thinking of potential failure modes in concept/design phase and process will enable early identification of failures. At these phases the possibility exists to choose new design/concepts/processes to eliminate the failure modes. This is of course highly favorable above finding solutions for problems found in the end phase of product development or even worse in the operational phase. Not thinking about possible failure modes in advance results in more problems found during test and integration, which will be experienced as the system not being robust enough. The result of this is that the product is being delayed for an undetermined period of time.

5.3.5 Applicability

Software FMEA must be performed as a pre-design step on each development level (system, sub-system, unit and module). The outcome of the Software FMEA will be used as design input. For the FMEA to have full effect it could also be done on a concept or architecture.

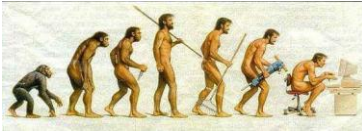
5.3.6 Structure

As stated earlier FMEA has been around for a long time and has been well documented. The difference between performing an FMEA on hardware is that for software the focus of the failure mode should not be the software itself but rather the environment in which it runs. This means focusing on e.g. timing fluctuations of other parts of the system or wearing or tearing of hardware like e.g. hard disc. The assumption is that software does not wear out and that a software bug should have been found during the test phase but for certain will behave consequently.

The outcome of the Software FMEA should be recorded in the next template (one example filled in):

Nr	Function block Process step	Potential Failure Mode	Potential Effect (s) of Failure	Potential Cause of Failure	Current Controls	P	S	*) w	D	RPN	Recommended action	Owner	Due date
1	Ethernet traffic	Defective optical giga bit Ethernet cable	Missing data on receiver side	Too tightly wrapped		3	5		5	75	Check link speed on network interface card	MC	2007-03-12

*) w = weight factor (to incorporate moment that the failure effect will be noticeable to customer), default [w=1]



<1 year [w=1.5]

1-2 year [w=1.2]

>2 year [w=1.0]

Software FMEA is performed in the following easy steps

- get acquainted with the system
- brainstorm on failure modes
- determine the effect of the found failure modes
- determine the risk of the failure mode
- determine the failure modes that must be addressed

5.3.6.1 Get acquainted with the system

A Software FMEA should be performed with persons that have a solid understanding of the system of which the software component is part, through well preparations.

5.3.6.2 Brainstorm on failure modes

Formulate a small group of persons with different views on the system and the software unit to create a list of failure modes. The brainstorm sessions should be kept should (should not exceed 30 minutes) to have maximum effect. Depending on the complexity of the system more brainstorm sessions could be needed.

An important aspect is that only one failure mode is addressed at a time. Various failure modes might have overlapping effects so it is important to specify them isolated per failure mode. Failing to do this will result in situations in which multiple failure modes are combined thereby losing the possibility to discriminate to the right failure mode.

It is the moderators' job to avoid discussions about e.g. effects and priority. These items will be addressed later in the process.

5.3.6.3 Determine the effect of the found failure modes

This part of the process must be done with persons that have indebt knowledge of how the software works in the system. Only they can determine the impact of a failure mode on software.

5.3.6.4 Determine the risk of the failure mode

The risk can be expressed as the product of change of occurrence, the change to detect the failure and the impact of the failure on the system.

$$\text{RPN}^* = \text{Probability} * \text{Severity} * \text{Detection}$$

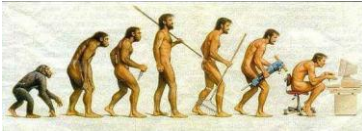
* RPN = Risk Priority Number

Appendix A has tables to differentiate between levels of change, detection and impact.

5.3.6.5 Filter the failure modes that must be addressed

This figure (Risk Priority Number) can be used to discriminate which failure modes to address and which to leave as is. The cost to mitigate (to implement and test) the failure mode must also be taken into account.

For the failure modes that will be addressed recalculate the RPN.



5.3.7 Participants / Collaborations

The following functions are involved in this pattern:

- Architects, designers, developers, manufacturing engineers, service engineers, suppliers and even customer/user can do the initial identification
- Managers (project managers) have to approve the cost and time to mitigate the failure modes. They also have to decide where the line is drawn on which failure modes should be solved.
- Moderator. This person does not need to have any knowledge about the context but is responsible to structure the meetings and keep focus during the meetings.

5.3.8 Consequences

The consequence of performing a Software FMEA is to spend more time on architecture, design and implementation. The return on investment is less integration time and a more reliable system with higher quality.

5.3.9 Implementation

When formulating a team to brainstorm about failure modes it might be useful to invite persons of other projects which have experience with some of the concepts as e.g. Ethernet or RAID configurations.

If the system under development is similar to an already available system it could be beneficial to perform root-cause analysis on the found problems and add those to the list of failure modes.

Try to isolate the failure mode during the brain storm session. E.g. packet loss on Ethernet can have miscellaneous root-causes. Cabling, network interface card, software stack, switch/router, etc are all failure modes.

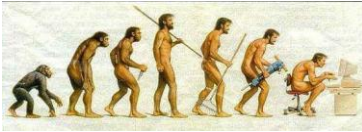
When the list of found failure modes is very long it could be useful to quickly reorder it so have implausible failure modes are at the end of the list. This way the less plausible ones can be skipped (but keep them recorded).

Lowering the RPN can be done by lowering the severity (e.g. a retry mechanism) or by adding measuring point so that the failure mode can be detected and reported clearly.

5.3.10 Sample

Ranking of Probability, Severity and Detection

value	Meaning:	ranking
Very high	Failure is almost inevitable	10
High	Repeated failures i.e. design new or corresponds to earlier designs, which had a high field call rate	8
Moderate	Occasional failures i.e. Design corresponds to earlier construction and showed to have low failures	5
Low	Relative few failures i.e. Design corresponds to earlier constructions and had no/ very low field calls	3
Remote	Failure is unlikely	1



Source: IEC 60812:2006, comments added based on CFT Hilberink

Table 2- 1. Probability Ranking (root cause in design)

	Severity of the effect (S)	ranking
Very high	Failure mode involving potential safety problems and/or conformance to federal regulations. It might endanger the operator or patient. (10 without warning, 9 with warning)	10
High	Serious customer dissatisfaction due to the nature of the failure effect such as a non-operational (part of a) system. It does not concern a safety issue or non-compliance to regulations.	8
Moderate	Causes some customer dissatisfaction or annoyance. The customer notices a deterioration of the working of the system. It might require reparation.	5
Low	The failure will only cause a slight user dissatisfaction or annoyance. The customer will possibly notice a slight effect on the working or performance of the system.	3
Minor	Unlikely that the failure will have a noticeable effect on the working of the product. The user will probably not notice the effect.	1

Source: Philips Display Components -scale modified by G.J. Laurensen

Note: Early failures (e.g. during warranty period) might create more annoyance to our customers. Therefore the introduction of a weighting factor might be considered. This can be incorporated as an additional column in the FMEA table.

Table 2- 2. Severity Ranking (effect on end-user)

	meaning	ranking
Absolutely uncertain	Design control will not and/or cannot detect a potential cause / mechanism and subsequent failure mode; or there is no design control	10
Remote	Remote chance the design control will detect a potential cause / mechanism and subsequent failure mode	8
Moderate	Moderate chance the design control will detect a potential cause / mechanism and subsequent failure mode	5
High	High chance the design control will detect a potential cause / mechanism and subsequent failure mode	3
Almost certain	Design control will almost certainly detect a potential cause / mechanism and subsequent failure mode	1

Source: IEC 60812:2006

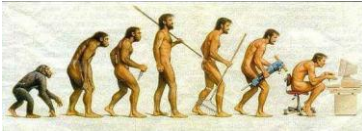
Table 2- 3. Detection ranking (timely detection of a failure)

5.3.11 Known uses

Philips Medical Systems

5.3.12 Related patterns

Root cause analysis pattern



5.4 Security and Privacy pattern

5.4.1 Pattern Name and Classification

The Security and Privacy pattern ensures the confidentiality, integrity and availability of assets directly or indirectly involved with a product. The asset definition depends on the product but in this case consists of items such as personal data, intellectual property, system applications, configuration parameters, etc. This is an aspect of development that is relative new and also important to incremental and evolutionary software development.

5.4.2 Intent

The intent of this pattern is to

- Detect, analyze, document and if possible mitigate security and privacy risks as soon as possible during the development process or when released during the entire period that the product is under support.
- Security and privacy flaws are risks to the end user, third parties and the manufacturer of a product.
- Ensure that a product complies with privacy and other legislation to ensure, within business acceptable boundaries, that (un)intended misuse is unlikely.

5.4.3 Also Known As

Security and Privacy are specializations of risk analysis, requirements gathering, code analysis, testing and validation and last but not least part of the life cycle management process.

5.4.4 Motivation

- It is important to protect customers or third parties of our products against threats like identity theft, misuse of banking accounts and disclosure of other personal data. Not only because it is mandated by law, but also because such a breach can seriously damage a company's reputation and induce a high cost if breach notification is mandated by law.
- Managing security and privacy of risks to a company also falls under compliance such as the Sarbanes-Oxley Act because an incident can have major financial repercussions.

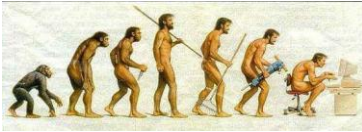
5.4.5 Applicability

- This pattern is applicable to all process models.
- Performance can be measured using code reviews, code analysis tools, security scanning tools, compliance to security/privacy requirements, number of incidents and the results from a dedicated penetration test team.
- Security scanning tools exist on source code level and application / OS level.

5.4.6 Structure

This pattern enforces actions to be addressed at several stages of the development process:

- During project definition a security and privacy risk analysis should be performed to surface the vulnerabilities as earlier as possible during the development cycle. A risk is the product of the severity of the vulnerability against the likelihood that it can occur. These risks should be addressed by the business whether they are acceptable, if not enforced by law, or should be mitigated.
- During project development the development team should perform code reviews specifically focusing on security and privacy issues. This can be done on the entire code



- or, depending on the defined business level, only on essential assets as defined by the initial security and privacy risk analysis.
- During project development and sub-component deliveries several tests are performed each time to ensure that no new vulnerabilities are introduced.
 - During final project test and verification a test team performs security scans of the product to document the current state and to validate that the mitigations, as approved during the initial security and privacy risk analysis, have been resolved and to surface possible new issues. The final state and findings are document in an updated security and privacy risk analysis report that is signed by the product manager, development manager and product security officer.
 - As a corporate level audit it might be essential to hire a specialized outside firm to perform black or white box penetration testing to validate the works of the development team.
 - When a product is released it should be monitored during its entire commercial life to ensure that new vulnerabilities discovered in the product or components used by the product do not break the determined level of confidentiality, integrity and availability. This work should be done of a dedicated team of security/privacy savvy personnel.

5.4.7 Participant

Project management processes
Other risk related processes such as safety and business risks
Requirements management
(Peer) review processes

5.4.8 Collaborations

There are several levels of collaboration each with their own level of expertise. The highest level is the corporate Product Security Leadership Council that defines the security and privacy policies for the organization. Next level consists of security specialist within the project teams. Security awareness should be assured in the entire development group including architects, designers, implementers and testers. This is achieved through presentations, training and participation in security / privacy analysis. Informing management of state, progress and incidents is also an important step to ensure overall acceptance.

5.4.9 Consequences

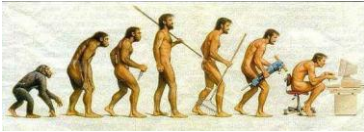
This pattern supports and proves the objectives by:

- Ensure requirements are set and validated
- Reporting at start and end of the project
- Early detection of possible issues

5.4.10 Implementation

There are several security and privacy risks analysis methods and tools for the office IT environment. They do not directly match on a security and privacy risks analysis for products since the environment in which the product will live is uncontrolled. An example of such a method is OCTAVE. Parts of this method are used during our product security and privacy risks analysis.

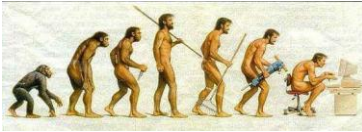
Enforcing security and privacy mandates the support of management and corporate policies. Although the business risks on a top management level might be quite obvious the product



marketing teams that drive product requirements need this extra trigger since these requirements are often stated as 'not demanded by our customers'.

Two main policies drive the implementation into the organization. One mandates the security and privacy process while the other policy defines the requirements that should be implemented by all products.

First step is to setup a specialized team to start driving security and privacy into the organization. When this is setup this team should handover the work to the normal development teams. This starts with awareness follows by the appropriate training on all levels, e.g. OS hardening, risk analysis, data classification, secure coding, etc.



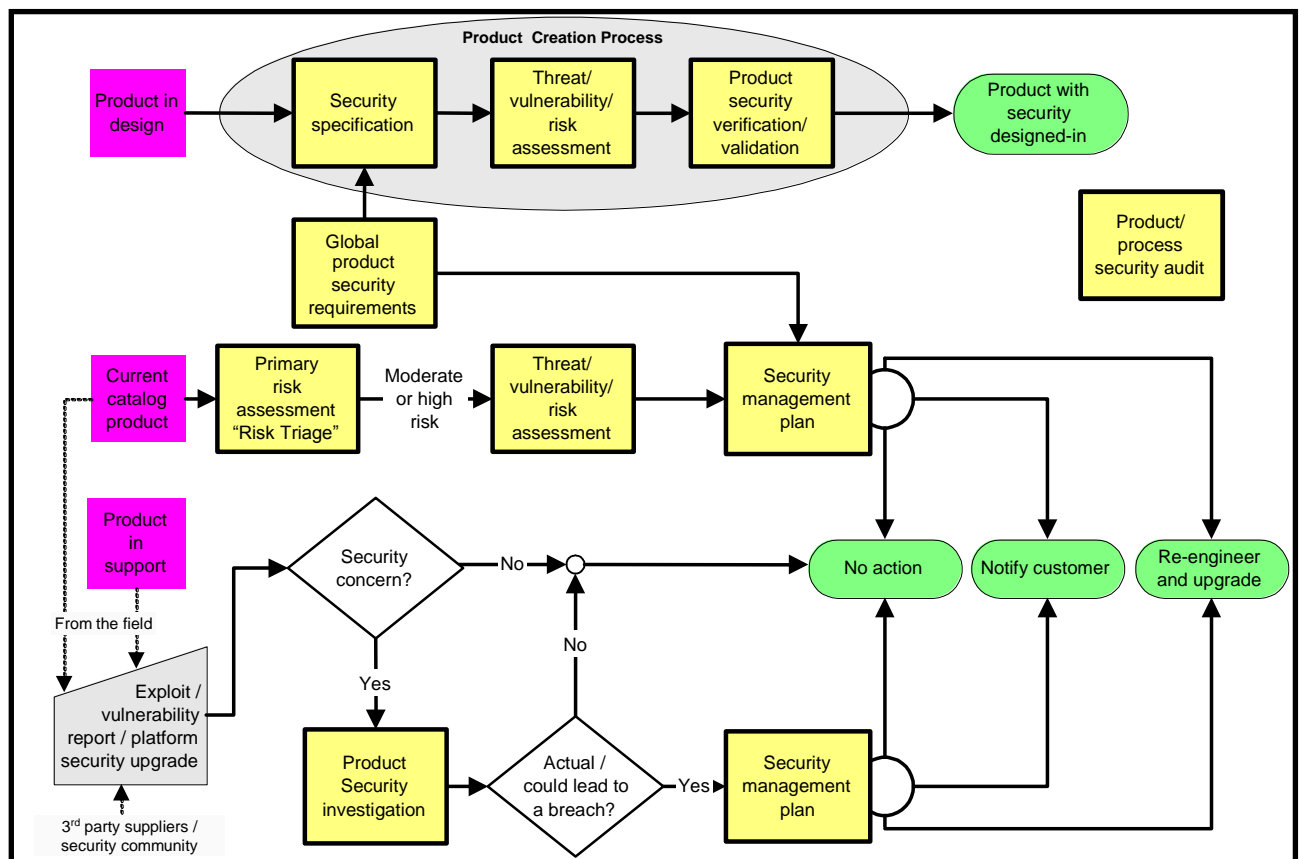
5.4.11 Sample

The security and privacy process is applicable from the start, e.g. conceptual phase of a product right up to the end of support point in time. It thereby overlaps all processes within the company that involve development, support and life cycle management.

When using this process there is distinct difference between the type or state of a product being either a product in design, a product currently in the catalog and being shipped or a product that is no longer sold but still under support. The following figure shows the difference in the treatment of the three different type or state of a product to address security and privacy.

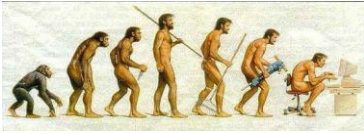
This picture does not clearly indicate that the product security verification and validation is performed multiple times during the process creation since this is an iterative process. Also subsequent minor releases of the product or service packs will undergo these tests.

Activities in rectangular boxes produce documentation and are specified by explicit processes under the product security and privacy policy.



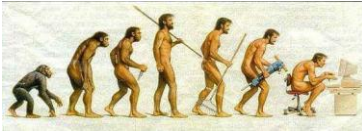
5.4.12 Known Uses

Currently, this pattern is applied within Philips Medical Systems.



5.4.13 Related Patterns

None



6 Code related Realization Process Patterns

6.1 Continuous Builds Pattern

6.1.1 Pattern name and classification

In the **continuous builds** method, software builds are automatically created from the source code and tested every fifteen minutes. The method contributes to the quality of the software and the effectiveness of the development process by highlighting critical errors in the software early on.

6.1.2 Intent

Automated continuous builds creates a new build from the sources, runs a series of automated back box test and creates a report every 15 minutes. Would the build or the tests fail, the process creates an automated error message and sends it to the developer and the software architect in case whom can then address the issue immediately.

6.1.3 Motivation

Finding and correcting software errors as early as possible saves both time and money. Once a critical error gets to go unnoticed for a period of time, the amount of resources required to fix it can increase almost exponentially if the bug affects other parts of the software causing multiple people fixing multiple errors. In the case of multiple software programs interacting with each other, the case is even worse.

6.1.4 Applicability

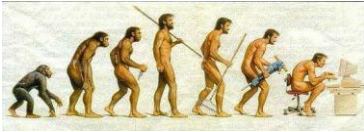
The general method is applicable in all kinds of software development projects. However, Calassa Labs' implementation of the method is only suited for rather small development teams as currently there are no automated tools for solving cases where more then one developer has checked in erroneous code simultaneously. These situations still require co-operation between the developers and the architects.

6.1.5 Participants

The participants are the software architects and developers using the method in the development work and creating and maintaining the tests.

6.1.6 Collaborations

Once an error is detected, the system sends an automated error report to the software developer and the software architect. In most cases, the software developer fixed the error and checks in new fixed code. If needed, the software architect assists the developer to address the issue. In the case there are several errors affecting the same build, the developers will coordinate their efforts in solving the issue.



6.1.7 Consequences

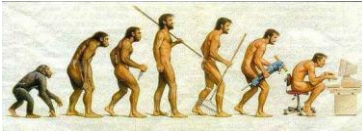
Being an agile method, the continuous builds method speeds up the development process and enhances the quality of the software whilst enabling the developers to better focus on the actual development work instead of creating builds, testing and fixing old errors.

6.1.8 Implementation

Due to the extensive work required to develop and maintain the required tests, the method is not suitable for all software projects. In the case of small, simple software projects, the implementation might require more resources than could be saved by utilizing the method. On the other end, with complicated software projects with multiple configurations and interactions, maintaining the test base could prove to be next to impossible.

6.1.9 Known Uses

Most companies committed to developing quality software will have similar methods in place. Also, commercial solutions are available to implement automated builds and testing.



7 Testing related Process Patterns

7.1 Risk based Testing pattern

7.1.1 Pattern name and classification

Risk Based Testing

7.1.2 Intent

The purpose of risk based testing is to divide the limited amount of test time and resources such that the total costs of defects are minimized by spending most effort on finding serious defects. To be used to prioritize test effort based on technical and business risk analysis. This is necessary within evolutionary product development to find the most important defects as early as possible within given time boundaries.

7.1.3 Also known as

Product Risk Analysis

7.1.4 Motivation

Testing the (integrated) product is the last thing done in a project. In general, testing is always under pressure. A 100% test coverage is simply too costly. The benefits of Risk Based Testing are:

- Focus to detect the more serious defects.
- Test depth depends on risk level.
- At any given time, the test team can inform management clearly on the remaining risks.

7.1.5 Applicability

Risk Based Testing can be applied when other methods of organizing the test effort demand more time or resources than can be afforded. Typical situations are:

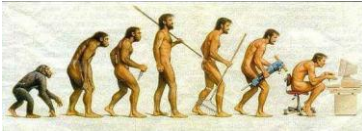
- Time-to-Market driven projects
- Technology driven projects with high uncertainties or that are difficult to plan
- Projects with complex software and hardware combinations.

In general, difficult to plan projects with high risks in combination with Time-to-Market pressure.

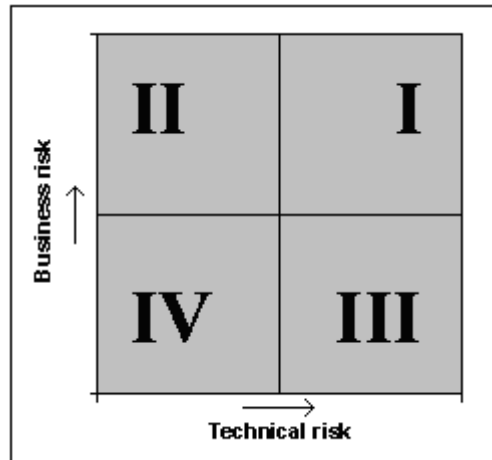
7.1.6 Structure

1. Involve stakeholders w.r.t. risks.
2. Make a prioritized list of risks.

The result is recorded in a Product Test Risk Matrix. This matrix is divided in four risk areas (quadrants I, II, III and IV), with the business risk along the vertical axis and the technical risk



along the horizontal axis.



3. Perform testing that explores each risk.
4. As risks evaporate and new ones emerge, adjust test effort to stay focused on the current crop.

7.1.7 Participants

At least the following roles participate in Risk Based Testing: Architect (s), Application representative(s), End user representative(s), Tester(s), Test Manager, Marketing representative(s).

7.1.8 Collaborations

Inputs for risk based testing are:

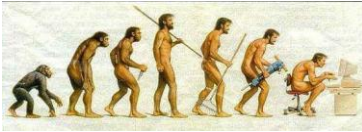
1. The technical aspects of the system.
These aspects are provided by the technical people of the project (architect, testers).
2. Use of the system.
Usage information is a/o provided by the customer and the end user.
3. Business risks analysis

Participants further collaborate by attending 2 meetings:

1. Kick-off meeting.
In this meeting, the correctness and completeness of selected test items, attributes, weight factors and stakeholders is verified. The rules are explained according to which the risk scoring has to be performed. Agreements are made about the assignment of attributes to the various stakeholders.
2. Risk consolidation meeting.
In the risk consolidation meeting with all stakeholders, an agreed Product Test Risk Matrix is defined. The test items with large differences in risk scores are discussed and corrected. If necessary, weight factors between the attributes are discussed and corrected. Furthermore, test items close to the border of two risk areas are discussed and corrected.

7.1.9 Consequences

- Eases the communication
- Remaining risks are clear at any time
- Serves as input for test strategy and test plan
- Ensure stakeholder involvement upfront
- Allows for differentiation in test approach:
 - Lightweight test techniques for low risk items



- Heavy weight test techniques for high risk items

7.1.10 Implementation

This input is based on the running projects within Philips Medical Systems.

Requirements Risk Matrix

par	function	pizza item	rank	par	function	pizza item	rank
II	9999 XXXXXXXX	XXXXXXXX	150	business risk	9999 XXXXXXXX	XXXXXXXX	169
	9999 XXXXXXXX	XXXXXXXX	150		9999 XXXXXXXX	XXXXXXXX	167
	9999 XXXXXXXX	XXXXXXXX	144		9999 XXXXXXXX	XXXXXXXX	163
	9999 XXXXXXXX	XXXXXXXX	138		9999 XXXXXXXX	XXXXXXXX	154
	9999 XXXXXXXX	XXXXXXXX	133		9999 XXXXXXXX	XXXXXXXX	154
	9999 XXXXXXXX	XXXXXXXX	127		9999 XXXXXXXX	XXXXXXXX	142
	9999 XXXXXXXX	XXXXXXXX	125		9999 XXXXXXXX	XXXXXXXX	142
	9999 XXXXXXXX	XXXXXXXX	125		9999 XXXXXXXX	XXXXXXXX	142
	9999 XXXXXXXX	XXXXXXXX	125		9999 XXXXXXXX	XXXXXXXX	140
	9999 XXXXXXXX	XXXXXXXX	121		9999 XXXXXXXX	XXXXXXXX	140
	9999 XXXXXXXX	XXXXXXXX	117		9999 XXXXXXXX	XXXXXXXX	129
	9999 XXXXXXXX	XXXXXXXX	117		9999 XXXXXXXX	XXXXXXXX	129
	9999 XXXXXXXX	XXXXXXXX	108		9999 XXXXXXXX	XXXXXXXX	127
	9999 XXXXXXXX	XXXXXXXX	98		9999 XXXXXXXX	XXXXXXXX	123
	9999 XXXXXXXX	XXXXXXXX	98		9999 XXXXXXXX	XXXXXXXX	123
IV	9999 XXXXXXXX	XXXXXXXX	100	technical risk	9999 XXXXXXXX	XXXXXXXX	125
	9999 XXXXXXXX	XXXXXXXX	94		9999 XXXXXXXX	XXXXXXXX	121
	9999 XXXXXXXX	XXXXXXXX	92		9999 XXXXXXXX	XXXXXXXX	106
	9999 XXXXXXXX	XXXXXXXX	92		9999 XXXXXXXX	XXXXXXXX	102
	9999 XXXXXXXX	XXXXXXXX	88		9999 XXXXXXXX	XXXXXXXX	92
	9999 XXXXXXXX	XXXXXXXX	88		9999 XXXXXXXX	XXXXXXXX	90
	9999 XXXXXXXX	XXXXXXXX	85		9999 XXXXXXXX	XXXXXXXX	90
	9999 XXXXXXXX	XXXXXXXX	83		9999 XXXXXXXX	XXXXXXXX	90
	9999 XXXXXXXX	XXXXXXXX	81		9999 XXXXXXXX	XXXXXXXX	73
	9999 XXXXXXXX	XXXXXXXX	81				
	9999 XXXXXXXX	XXXXXXXX	73				
	9999 XXXXXXXX	XXXXXXXX	65				
	9999 XXXXXXXX	XXXXXXXX	63				
	9999 XXXXXXXX	XXXXXXXX	63				
	9999 XXXXXXXX	XXXXXXXX	60				
	9999 XXXXXXXX	XXXXXXXX	56				
	9999 XXXXXXXX	XXXXXXXX	50				
	9999 XXXXXXXX	XXXXXXXX	50				
	9999 XXXXXXXX	XXXXXXXX	48				
	9999 XXXXXXXX	XXXXXXXX	48				
	9999 XXXXXXXX	XXXXXXXX	42				
	9999 XXXXXXXX	XXXXXXXX	19				

Definition of quality characteristics

Functionality

The capability of the product to provide functions that meets stated and implied needs when the system is used under specified conditions.

Reliability

The ability of the system to perform its required functions under user conditions for a specified period of time, or for a specified number of operations.

Image quality

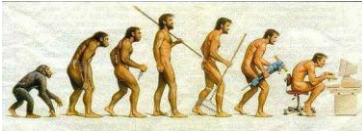
'The basic image quality' requirements for Cardiac and Vascular systems are defined in [SRS-IQ]. This document describes the system in terms of image quality parameters in such a way that if a system meets the requirements in this specification the image quality of that system is guaranteed. Note that IQ must be regarded in combination with dose management.

Interoperability

The capability of the software product to interact with one or more specified components or systems.

Usability

The capability of the product to be understood learned, used and attractive to the user when used under specified conditions.



Serviceability

The ability to diagnose and solve a problem easily, adequately and quickly. Most important here is the MTTR, the mean time to repair because of its influence on the system up-time

Manufacturability

The ability to actually manufacture the product. (E.g. are the requested tolerances feasible, is yield acceptable, are parts duplicable, are subcontractors available, production facilities in place, is the system producible in isolated manufacturing steps etc.) This ability too is primarily to be taken into account in the system design by the architects.)

Note: install ability is part of serviceability.

Security

Product Security includes all aspects of securing the Confidentiality, Integrity and Availability of data and systems in the healthcare environment.

Risk Management

The capability of the product to achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified context of use.

Norm compliance

The capability of the product to adhere to legal regulations related to its use.

Mapping of test techniques to quality attributes and risk quadrants

Quality attribute	I	II	III	IV
Functionality	Equivalence Partitioning Boundary Value Analysis Cause Effect Graphing Elementary Comparison Test State Transition Test (1 switch) Process Cycle Test (2 test m.) Use cases	Equivalence Partitioning Process Cycle Test (1 test m.) Use cases Exploratory test	Equivalence Part Boundary Value Analysis State Transition Test (0 switch) Use cases Real-Life Test ¹ Exploratory test	Use cases Real-Life Test ¹ Syntax Test Error Guessing Exploratory test
Reliability	Real-Life Test ¹	Real-Life Test ¹ Exploratory test ²	Real-Life Test ¹ Exploratory test ²	Syntax Test ³ Real-Life Test ¹ Error Guessing Exploratory test ²
Image quality	Equivalence Partitioning Use cases Real-Life Test	Equivalence Partitioning Use cases Real-Life Test	Equivalence Partitioning Use cases Real-Life Test	Use cases Real-Life Test
Interoperability	Equivalence Partitioning Use cases	Equivalence Partitioning Use cases Exploratory test	Equivalence Partitioning Use cases Exploratory test	Use cases Syntax Test Error Guessing Exploratory test
Usability	Process Cycle Test (2 test m.) Use cases	Process Cycle Test (1 test m.) Use cases Exploratory test ⁴	Use cases Exploratory test ⁴	Use cases Syntax Test Error Guessing Exploratory test ⁴
Safety & norm compliance ⁵	Elementary Comparison Test Process Cycle Test (2 test m.) Use cases	<not applicable>	<not applicable>	<not applicable>
Serviceability	Real-Life Test	Real-Life Test	Real-Life Test	Real-Life Test

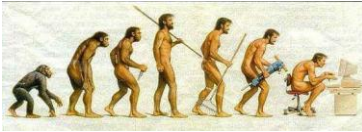
¹ Note that for Real-Life Test the current SRS and FRS specifications are not sufficient. We lack information about usage of the system in the field (how frequent/often are functions used, what functions are used most, what characteristics specific applications have, what characteristics specific users have, etc...). We need input from the User Profiles (or operational profiles) project.

² Exploratory testing is not merely a test design technique, but a test method. It is aimed at finding faults quickly. This kind of testing is not very suitable for reliability testing because we do it by using PTT/SE (or TAF) tooling in night batches (in spite of [TMM-TT]).

³ In contrary to [TMM-TT] is Syntax Test not suitable for reliability tests because we do it by using PTT/SE (or TAF) tooling in night batches

⁴ Exploratory testing is not merely a test design technique, but a test method. It is aimed at finding faults quickly. However the basics of the method can be used for usability tests when focus is on system usage rather than on finding faults

⁵ The indicated hazards and norm compliances always needs to be verified.



Manufacturability	Real-Life Test	Real-Life Test	Real-Life Test	Real-Life Test
Security	Process Cycle Test (2 test m.)	Process Cycle Test(1 test m.) Exploratory test	Exploratory test	Error Guessing Exploratory test

7.1.11 Samples

See above

7.1.12 Known uses

Currently, this pattern is applied in Philips Medical Systems.

7.1.13 Related patterns

All other test related patterns.

7.2 Incremental Testing pattern

7.2.1 Pattern name and classification

Incremental testing.

Incremental testing (and development) is a phased test approach based on identified risks in development implementation and testing. The Master Integration Diagram (MID) serves as a visualization of the available building blocks and the subsequent integration levels in time. It helps:

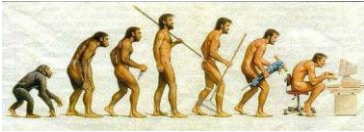
- Avoid inefficient big bang integration & test
- Mitigate risks at an early stage in the project
- To have measurable quality confidence (earned value) during project execution
- Improve timing predictability
- Improve product stability & quality.
- Possibility to deliver in stages to internal customers (system integrators)
- To have more uniform test effort capacity
- Result oriented high level project planning and tracking
- Clear overview of dependencies between building blocks and visualization of deliverables from 3rd parties

7.2.2 Intent

The intent of this process pattern is to maximally support the “integration is leading” approach in project oriented product realization activities and to provide optimal visibility of the defined integration steps and integration levels in time.

In the MID it is possible to identify what depends on what. In the sample below you can see the layered structure of the MID. The sample defines at the lowest level the development components. The development components integrate with the test environment, test cases are executed and the final delivery is a product with test results.

Note:



It is not intended to provide customers (or next in line organizations that add value) with evolutionary functionality although if required for design-in purpose it can be agreed to use an increment as intermediate delivery.

In that case the advantage of early stable tested versions of the product, although not complete, is that the (internal) customer can start (early) integration as well. Instead of delivering every week a not tested unstable version to the customer it is now possible to deliver at predefined moments a stable version to the customer.

7.2.3 Also known as

Generally the methods Cyclic Development, Evolutionary development, incremental development are strongly related. However the focus is here on Integrations & Testing

7.2.4 Motivation

Problem description:

Multi-discipline product developments often lose valuable time and effort as a result of late integration of developed components. This is mainly caused by decomposition of work products to isolated development disciplines and late integration of those work products into the end product. Integration issues and defects found in this stage are costly, time consuming (root-cause analysis is more difficult) and result in rework and time slippage. On top of that the responsibilities for integration activities are often not clear. (I.e. performing HW/SW integration and acceptance testing of 3rd party deliveries)

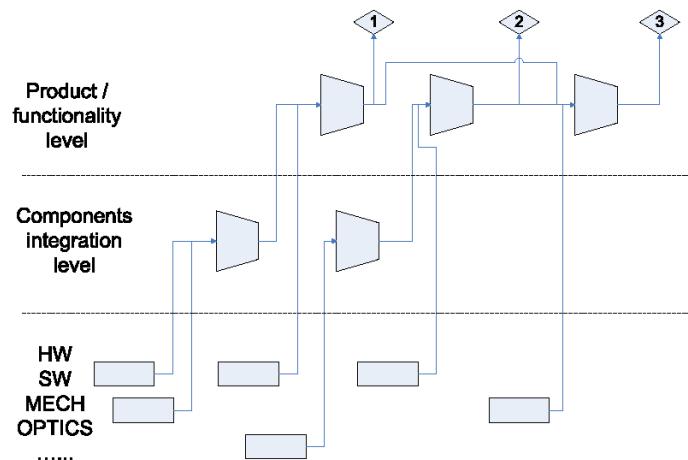
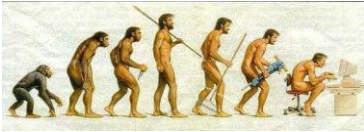
The incremental testing approach and the MID deal with this undesired phenomenon by defining manageable integration steps and tested increments that represent already parts of the functionality required in an early stage.

7.2.5 Applicability

The pattern is started during the project definition stage and applicable during the realization/implementation and test phase of product realizations.

7.2.6 Structure

The picture below shows the essence that this process pattern will provide.



The process steps of this pattern:

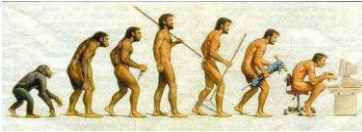
- Set-up
 - Define the product components and use these components in the next steps.
 - Make a first draft based on the high-level requirements and product decomposition.
 - Perform a components availability and test oriented risk estimation session
 - Set-up the MID starting with the high-level risk items
- Approve
 - Review MID (test manager + project manager + system designer, for larger projects team leaders and HW/SW designer as well)
- Monitor
 - Monitor the progress during weekly progress meetings, update with actuals of team schedules
 - Monitoring is performed on short term as well as long term activities.
- Change Control
 - Discuss changes needed during the progress meeting and adapt the MID accordingly.
 - Changes can come from two sides: 1) Product change requests, 2) Re-planning

7.2.7 Participants & collaborations

The following functions are crucial for this pattern:

- Architect/Senior Designers: Defines
- Test Manager (Integration Manager): Defines
- Project manager: Efficient, End result Risk Defines the number of increments together with the architect

Architect, Project Manager and Test Manager must cooperate well to ensure all pieces of the puzzle fit well together: they must define the Integration steps in advance and discuss impact of changes during the course of the project together.



7.2.8 Consequences/pre-requisites (goal/results)

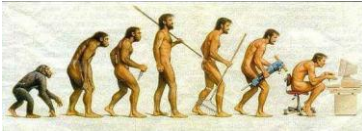
The main goal of this pattern is to have an efficient, fast, controlled product development process by defining clear integration points throughout the development life cycle. As a consequence, the project (test) team can always have a tested product base line at their disposal and therefore can ensure quality of the final product from the very first increment onwards. Also, if needed, the decision can be taken to release a certain increment, without having to wait for the next increments to finish. (eg. respond fast to market changes).

7.2.9 Implementation

Aspects to be taken into account for this pattern are:

- The MID is not the equivalent nor replaces the project (task) schedule.
(i.e. Microsoft project planning)
Individual tasks and milestones still need to be planned and tracked in a planning tool. Coupling, either manual or automatically) the MID and the project schedule at certain predefined increment positions will make the use of the MID even more powerful.
- Assign critical integration activities to competent people in the project and monitor these activities in a separate meeting.
- Suggested tools used are MS Visio or MS Excel (easy coupling with project schedule)
- The MID can also be used to calculate test system needs. It provides insight when there is a peak in the amount of test systems needed.

7.2.10 Sample



for all kinds of deliverables (documents, prototypes, models, source code, libraries, etc...). This process pattern describes how to organize, conduct, and follow through on the review of one or more deliverables.

7.3.2 Intent

The Technical Review Pattern describes a methodic way to assess the quality of your deliverables and also to ensure that what you deliver meets the needs of your customers. All the actors involved in the review process (reviewers and *reviewees*) are aware of the details of the process and therefore have a well defined role in it.

The outcome of this process pattern is basically a document including recommendations for overcoming the weaknesses detected in the deliverable.

7.3.3 Also known as

No other name available yet.

7.3.4 Motivation

Problem description:

There is a big amount of deliverables to be submitted during the development/maintenance of a SW product or module. The quality of these deliverables will determine the final quality of the product. Moreover, it often happens that some of the deliverables rely on previous ones, and if any early deliverable is not properly finished, it will worsen the quality of those relying on it. It is a proven reality that the cost of fixing defects increases the later they are detected in the development cycle. This process pattern helps detecting errors early. Besides, using Technical review implies handing your job to third parties that are experts in the matter: this provides some external “pair of eyes” to review your deliverable and you get to communicate your work to others, keeping the team informed of what is going on.

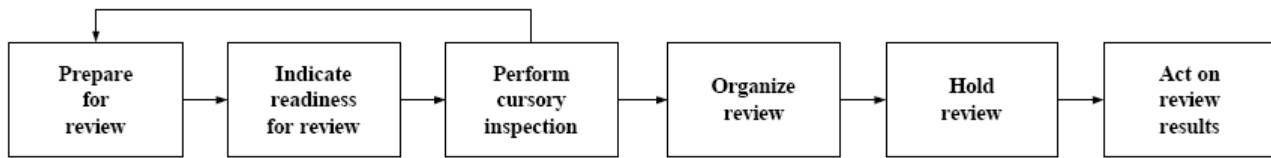
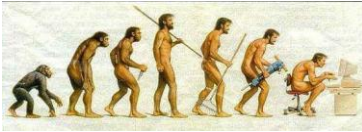
7.3.5 Applicability

The pattern is applicable throughout the complete development lifecycle of software development.

7.3.6 Structure

The initial context for this process pattern to be applied is: there are one or more deliverables to be reviewed, the deliverables are ready to be reviewed, and the development team is ready to have the deliverables reviewed.

The picture below shows the different steps involved in this process pattern



1. **The development team prepares for review.** The item(s) that are to be reviewed are gathered, organized appropriately, and packaged so that they may be presented to the reviewers.
2. **The development team indicates that they are ready for review.** The development team must inform the review manager, often a member of quality assurance, when they are ready to have their work reviewed as well as what they intend to have reviewed.
3. **The review manager performs a cursory review.** The first thing that the review manager must do is determine if the development team has produced work that is ready to be reviewed. The manager will probably discuss the development team's work with the team leader and do a quick rundown of what they have produced. The main goal is to ensure that the work to be reviewed is good enough to warrant getting a review team together.
4. **The review manager plans and organizes the review.** The review manager must schedule a review room and any equipment needed for the review, invite the proper people, and distribute any materials ahead of time that are needed for the review. The potential contents of a review package are discussed in the next section.
5. **The review takes place.** Technical reviews can take anywhere from several hours to several days, depending on the size of what is being reviewed, although the best reviews are less than two hours so as not to overwhelm the people involved. The entire development team should attend, or at least the people responsible for what is being reviewed, to answer questions and to explain/clarify their work. There are typically between three to five reviewers, as well as the review manager, all of whom are responsible for doing the review. It is important that all material is reviewed. It is too easy to look at something quickly and assume that it is right. It is the job of the review facilitator to ensure that everything is looked at, that everything is questioned.
6. **The review results are acted on.** A document is produced during the review describing both the strengths and weaknesses of the work being reviewed. This document should provide both a description of any weakness, why it is a weakness, and provide an indication of what needs to be addressed to fix the weakness. This document will be given to the development team so that they can act on it, and to the review manager to be used in follow-up reviews in which the work is inspected again to verify that the weaknesses were addressed.

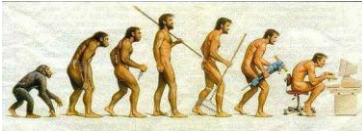
7.3.7 Participants & collaborations

The following participants are crucial for this pattern:

- Development Team: people developing the deliverables to be reviewed.
- Review Team: people in charge of reviewing the deliverables
- Review Manager: person coordinating the Review Team

7.3.8 Consequences

The main goal of this pattern is to assure the quality and suitability (in terms of meeting user needs) of a SW development by validating all the intermediate deliverables. This way, Senior management is assured that the development team has produced quality deliverables that meet



the needs of their user community. The development teams, and the reviewers, have a better understanding of the deliverables that they are building and how their work fits into the overall software project. Individual team members and reviewers are likely to learn new techniques during the review, either techniques applied to the deliverable itself, management techniques applied during the review, or development techniques suggested during the review to improve the deliverable.

7.3.9 Implementation

When implementing the pattern, be aware of focussing on reviewing the **content** of the the workitems that are being reviewed. Do not apply the pattern mechanistic, but really try to find the shortcomings or problems of a work item.

Plan reviews, just you would plan other activities of a project. If not, the project team may not have enough time for reviewing.

7.3.10 Sample

Not applicable, no additional information besides the “Structure” paragraph of this patterns

7.3.11 Known Uses

Used within Ibermatica, Philips and Alcatel

7.3.12 Related Patterns

None

7.4 Verification & integration in incremental development pattern

7.4.1 Pattern Name:

Verification & integration in incremental development: Integrating delta functionality on existing systems

7.4.2 Intent & Motivation

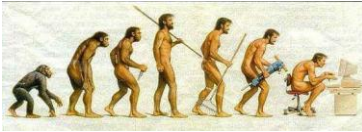
In incremental system development new developments are based on existing system platform and system functionality. The latest released HW platform and base lined SW archive (final release) is the starting point for new developments. As the latest platform is in general delivered to customers there is in general a commercial request to deliver upgrade packages of the newly develop functionality as well as initial deliveries. The need for upgrades has an impact on the verification process of the product due to number of HW and SW configurations of the so-called installed base which need to be verified.

7.4.3 Also Known As

No other names known

7.4.4 Applicability

The process pattern is to be used in the integration and verification of incremental product developments



7.4.5 Structure

Four plans form the basis of the integration and verification process in the incremental development:

- Integration plan (IP)
- Master Test Plan (MTP)
- System Verification Specification (SVS)
- Supported Configurations Specification (SCS)

The **integration plan** (IP) describes all the activities related to the integration of components to subsystem or system level. In the plan selected configurations are described (test system), the date when component integrations will take place and the tests which need to be performed on the particular configuration.

The **master test plan** (MTP) describes the overall project test strategy, approach, limitations and test activities to guarantee a well-tested product. This plan is the basis for managing and tracking all test activities in the project. The contents of the IP and MTP differ little from the IP and MTP for initial developments. However due to the variety of configurations in incremental development regression testing is a key issue in the verification in incremental development.

The test strategy is based on several analyses:

- Where are the risks from a technological point of view?
- Where are the risks from a business point of view?
- What norms and standards compliance are required?

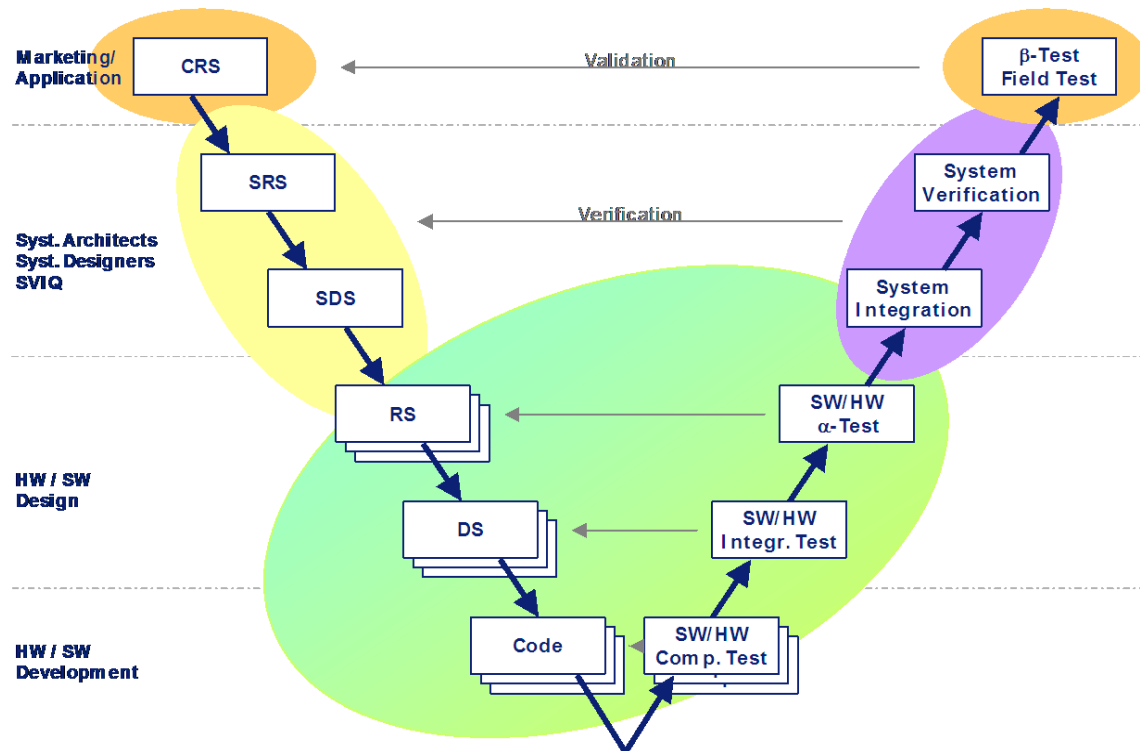
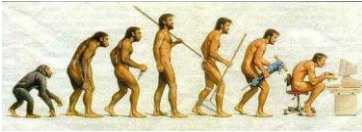
The **system verification specification** (SVS) specifies the system tests to be performed at system level. Usually the SVS is a total list of the system test specifications for specific functionality at system level e.g. safety verification and performance verification are part of the SVS list.

The **Supported Configurations Specification** (SCS) describes the system configurations which need to be supported by the new developments. The support of existing configuration is specific for incremental development as newly developed items can be used on platforms in the field. As the systems installed base grows over the years the number of to be supported configurations grows which has an impact on the verification effort to be done. The SCS is input for the risk analysis as described in par. integration /testing

V-model

The well known V-model is used as the model for initial as incremental developments, the main differentiator is the use of delta specifications in the requirement definition phase and the re-use of test specifications in the verification phase.

In the requirement definition phase the updating of the system requirement specification (SRS) is a substantial amount of work for complex systems. Even small updates of the SRS require overhead which might result in case of frequent updates in an-workable amount of overhead. The delta specification speeds up in the initial phase because it supports discussion and decision making. The delta specification needs however to be incorporated in the SRS before the start of the verification phase. It is to say the SRS needs to be authorized at the end of the design/implementation phase.



7.4.6 Participants and collaborations

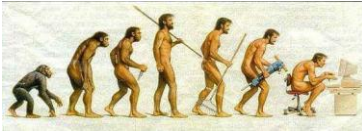
The following people play a role in the integration and verification with incremental development.

- System architect writes the delta system requirement specification and merges the requirements in the SRS before the verification phase. The system architect is also owner of the supported configuration specification.
- Test manager writes the system verification specification and is chairman of the Development Integration Team (DIT). This team controls the system integration and defines the content of the different test system configurations. A project proposal is made by the DIT for the number of test systems which need to be allocated by the project manager. Estimation of verification effort (time, resources) is done by the test manager on the content of the SRS (functionality) and SCS.
- Project manager and sub project managers use the delta SRS, SCS for input and effort estimates.

7.4.7 Consequences

Fade in / Fade out operations

As the new developments are usually based on the platform which is currently in production the factory introduction of the newly tested release has an impact on the fade-in / fade-out between the new and previous release. At the end of the verification phase systems are allocated in the factory and build according the newly developed configuration. These so-called pre production models (PPM) will be delivered to the field however as a first of a kind, assistance of development on a regular basis is required to solve system problems. A special team with



representatives of development and factory has the assignment to control the PPM progress and solve problems which may arise during the building of the PPM.

Configuration control

For integration and testing we have to differentiate between SW and HW development. In general the SW development is based on an existing working archive which will be modified for the new release with new functionality and/or architectural changes. It has preference that after any SW integration step the archive is tested on its functionality and performance. This guarantees a stable archive and prevents big bang integrations where archive quality could be lost. This approach needs to be described in the test strategy (MTP) of the project. Maintaining the quality of the archive is key for incremental testing.

Integration / Testing

In an incremental development approach, a large number of components will be re-used from the previous product and a limited number of components will have to be changed.

The components are in general developed for the latest platform but could also be used for the installed base configurations. It has serious commercial benefits when newly developed items can also be used in the installed base.

The increase of the number of configurations increases also the number of tests, as functionality has to be tested not only on one configuration but also on all other existing configurations. As testing time and test resources are limited not all configuration can be tested in the available time. It is however not only time that impacts the verification also the availability of HW for building specific system configurations limits the verifications.

In exceptional cases system verifications are transferred to systems in the field as the configuration could not be build on the development site.

In order to judge the risk of not performing verification a risk analysis is made on the configurations whether a particular functionality has a high risk of failure on a specific configuration. If the chance of failure is low and the business impact of failure is low the verification is limited or not performed at all.

Special attention is given to safety related items. In figure 1 an example of a risk matrix, representing a graphical view of the risk analysis. The numbers represent certain functionality on configurations which is to be tested.

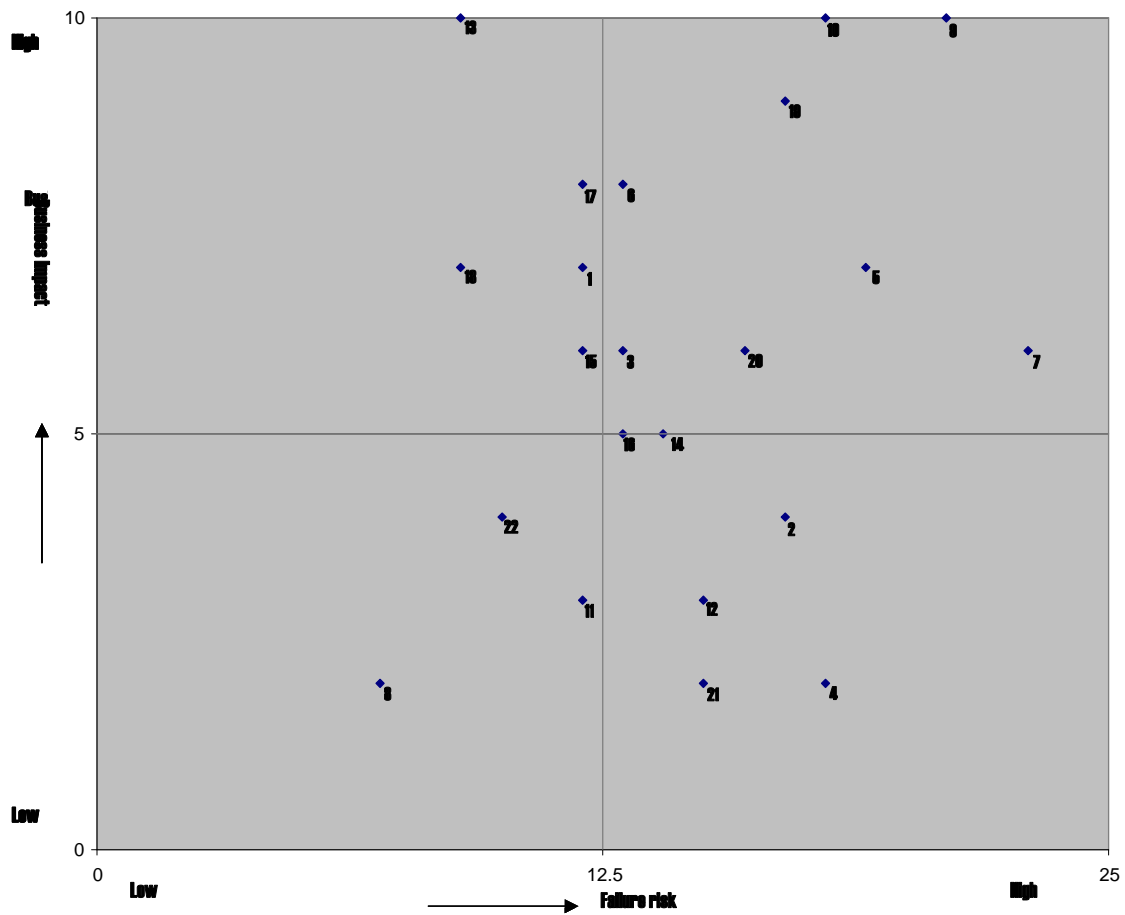
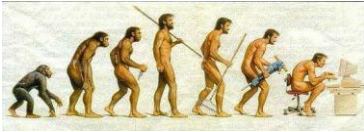


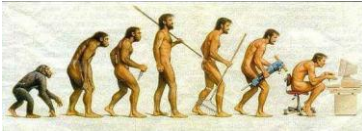
Figure 1 Test risk matrix

7.4.8 Known use

Philips Medical Systems

7.4.9 Related patterns

Delta specifications in evolutionary developments, Risk based testing, Incremental testing



8 Supporting Process related Patterns

8.1 Incubators for reducing project risk pattern

8.1.1 Pattern Name and Classification

Incubators for reducing project risk pattern

8.1.2 Intent

The incubator it is a best suitable process to reduce project risk regarding the time to deliver the first stable project release and to get the skilled people into the project team. Their applicability is intended in the first phase of the project lifecycle. Two main outcomes are expected from the incubator process:

- The establishing of the project: scope description, high level architecture description, defining of related projects, define initial milestones, etc.
- Create community: identify some key assets such as available initial code, people that will be initial committers, roles in the project.

Some assets addressed:

- Identifying better alternatives (to avoid duplicated effort)
- Provide a clear definition of the main features, and a high level architecture, define application domain, and identify other related projects.
- Estimating project time line (schedule)
- Engage the best people with required skills (define initial team)
- Manage people expectations and the expected technical impact of the project.

8.1.3 Also Known As

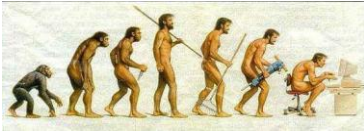
No other known names

8.1.4 Motivation

Currently there are many F/OSS projects, some of them have shaped big communities like ecosystems, but many others are founded in an isolated way, using the structures of collaborative development environments (SourceForge) as their platform; in this way these projects face the following risks:

- Not achieving critical mass of members (developers, committers, testers), therefore not conforming a sustainable community. If there, are no collaborative works (contributions), the project stops and even is cancelled.
- The project not delivering a stable release
- The project not producing any technical impact

Many risks that affect the software development project exists; the incubator is not a magic solution for these risks; there are other variables that can be taken into account such as:



principles of the companies, organizational culture, organizational structure (roles), processes and tools that support it, etc.

8.1.5 Applicability

It is applicable in early phases of the lifecycle and in more detailed way in the first phase (launch and establishment) when new projects are launched as part of mature projects; another way is as projects that cover requirements in new domains; in the above situations, the incubator pattern can be applied. Also suitable when the start point project is an existing code base.

The pattern offers four characteristics to be evaluated before making a decision about its application, thus:

- Organization (management)
- Culture principles
- Development process
- Tools and artifacts

Each organization has its own priorities, therefore a value scale must be defined according to them and thus a risk factor can be taken into account. Then the decision about the best way to apply the pattern can be taken.

8.1.6 Structure

The incubator is a process to help in the establishment and launch phases of the software projects, and it can be divided in the following phases:

- Pre-proposal
- Proposal
- Validation/review (iterative phase)
- Graduation from incubation

This process can be integrated as a single phase in the lifecycle development or as a previous step the actual development, in both cases it should be taken into account as the initial stage in the project development.

Pre-proposal and proposal phases help to define briefly: the project scope, high level architecture, some features of the project, related projects and recruiting of the people.

Validation/review phase can be described as an iterative stage where new functionality is added through development activity with the goal to achieve as soon as possible a stable release.

Graduation occurs when the stable release is produced and approved by the competent role. The time from proposal is approved to graduation is known as incubation time.

Pre-proposal and proposal phases would be considered as “**project launching**”; validation/review and graduation can be considered as “**project establishment**” (see figure 1).

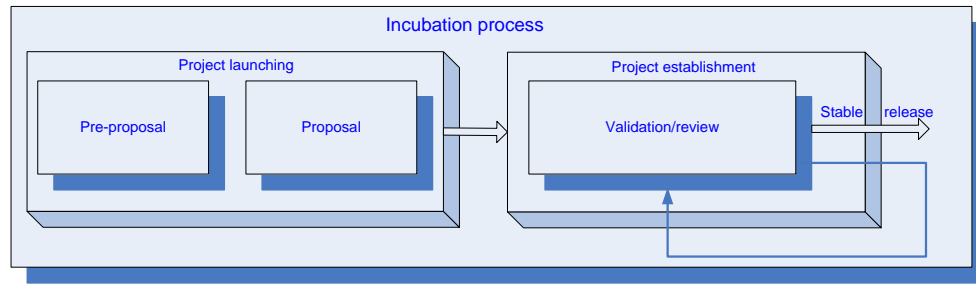
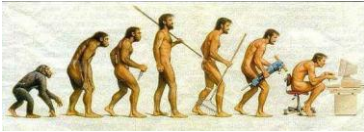


Figure 1. Incubation pattern

The entry required for the process:

- Code
- Features
- Requirements
- Related projects
-

Process outcome:

- Stable release
- Project team
- Experiences

8.1.7 Participants

A well defined structure of roles is required to establish the task, identifying the responsible role for each phase. As minimum desirable roles are:

- Incubator leader
- Committers (developers, tester)
- Customer (final user)
- Project leader
- Personnel from related projects

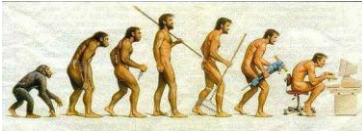
8.1.8 Collaborations

The key principles of the pattern are:

- The volunteer work and the collaborative way based on people skills. Thus the best way to define responsibilities it is to apply roles according to a meritocratic culture.
- The final user must be involved into the incubation phase, a leader that drives the process is required but the technical decisions (architecture, scope, new functionalities) should be voted by all (final user, operational people, leader).

Working in this way a stable release will be obtained as the project outcome.

8.1.9 Consequences



The main goal of the pattern is to support the establishment and launch of the project; in other words, avoid the risk of the project go inactive in this phase or risk of not achieve a technical impact. As the main consequences of the pattern applicability results:

- Proposal clearly defined: rational, main system features, high level architecture of the new system, system scope, related projects, initial code.
- Getting involved the most skilled people in the project
- Management peoples' expectations
- Delivering a stable release in short time

8.1.10 Implementation

Some aspects to be taking into account to implement the pattern:

- The existing structure, process of the lifecycle development
- The process must be the first stage of the lifecycle development
- Roles and organizational structure must be clearly defined
- Policies for decisions about termination or advancing project (optimum incubation time, size and skills of project team) should be in place.
- A progress monitor mechanism is required.

8.1.11 Known uses

Currently these patterns (process) are applied in [APACHE] and [ECLIPSE] that are both communities of Opens Source, but with some special characteristics as:

- Virtual and centralized structured
- Meritocratic culture (roles)
- System decision make by vote
- Collaborative and distribute development
- Limited resources
- Specific domain application

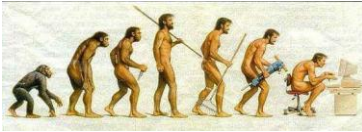
8.2 Incremental Configuration Management pattern

8.2.1 Pattern Name and Classification

Incremental configuration management

Incremental configuration management is a process that helps managing configuration management aspects of Software Components during incremental software development. It helps:

- Identify your configuration items in your products.
- Define & plan changes of these configuration items during an increment.
- Manage configuration items functional growth during & over increments.



- Improve product stability & quality.

8.2.2 Intent

The pattern describes optimal ways of working for managing your configuration management system when using an incremental development approach.

The outcome of this process pattern is:

- Well defined & recorded (traceable) configuration items
- It helps manage the quality level of configuration items during the increments of a project for the different stakeholders (e.g. developers, testers, external parties).
- Clear lifecycles during the incremental development of a configuration item (e.g. When changes start and stop)
- Possibility for multiple lifecycles of the same configuration item at a certain moment in time.
- Managed rules for changing the same configuration items across multiple increments (parallel development). Preventing that changes for one increment are lost when changing the same configuration item for another increment that runs partly in parallel.

8.2.3 Also known as

No other name available yet.

8.2.4 Motivation

Problem description:

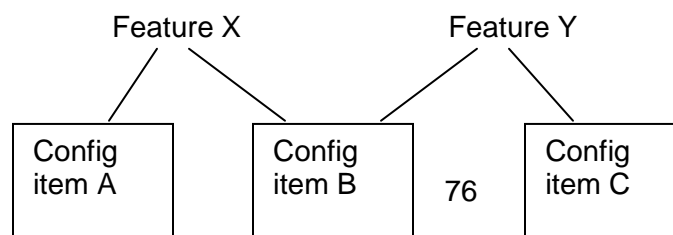
Incremental developments often struggle with unstable, and therefore inefficient working, software archives.

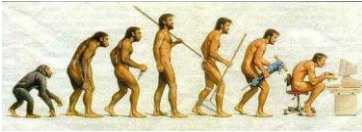
During incremental development changes are made to software. Most often this is software that already exists, and needs (functional) extensions; and sometimes we can start from scratch with an empty code base. The essence of incremental development is that each increment implements certain features and converges over time with respect to quality and delivers a stable product version. Convergences towards a stable software configuration item for one increment and at the same time making (large) changes to the same software configuration item for a next increment do not go hand in hand. This process pattern helps solve this problem.

A detailed example of the described problem is given below.

During the course of the (incremental) development a situation will occur that the same SW config item needs to be changed in more than one increment. The following example illustrates this and the problems that can arise:

Example: feature X is implemented by config item A and B. Feature Y is implemented by config item B and C. In increment 1 we want to deliver feature X, in increment 2 we want to deliver feature Y.





Increment 1: feature X

Increment 2: feature Y

For risk reduction or (development) timing constraints increment 1 and 2 partially overlap in time (e.g. increment 2 is starting while increment 1 is not yet finished). This means that in this example configuration item B is modified for increment 1 and at the same time needs modifications for increment 2.

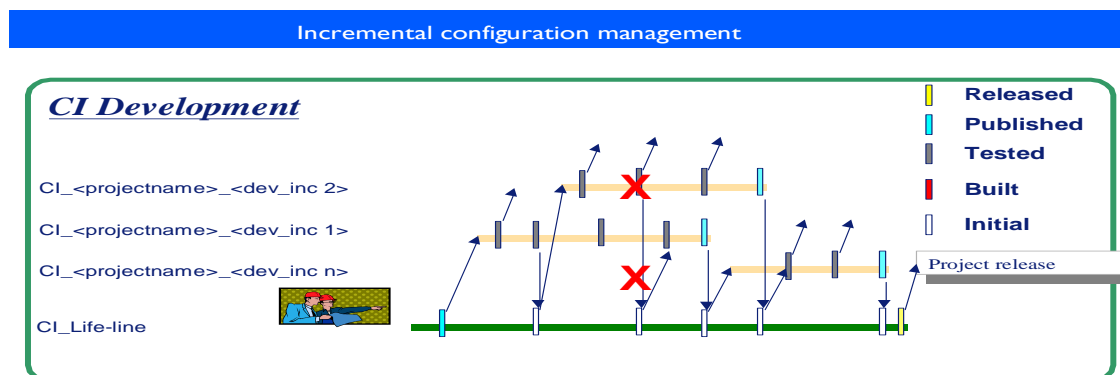
But increment 1 is coming to an end and a stable configuration item B is needed, so only urgent bugs can be fixed (minor changes), but no major (design) changes are allowed otherwise increment 1 will not converge to an end. So here we have a problem.

8.2.5 Applicability

The pattern is applicable throughout the complete development lifecycle of software development.

8.2.6 Structure

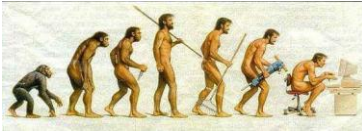
The picture below shows the essence that this process pattern will provide.



Remark: On the CI_life-line, increment synchronization **must** be **sequential in time**. increment 1 must merge back to life-line **before** increment 2.

The process steps of this pattern:

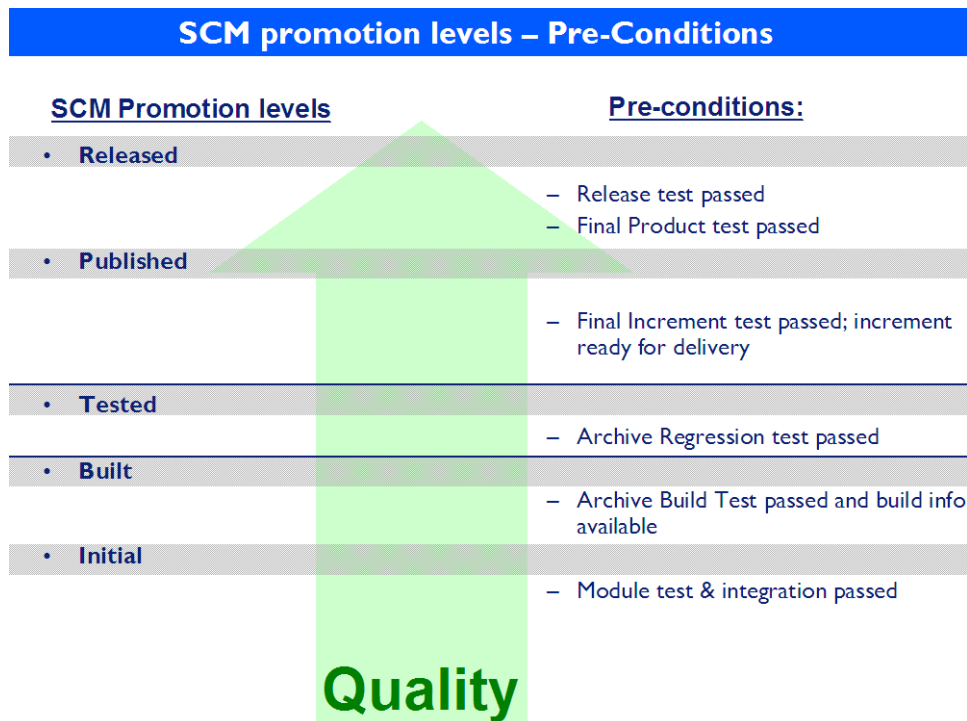
- At the beginning of a project, the Configuration Items (CI's) that this project will deliver must be clearly identified. Definition of a CI:
 - Controlled interfaces with respect to other configuration items.
 - “Own” documentation structure.



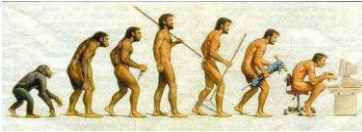
- Independent development, verification & validation.
 - “Own” life cycle.
 - Form, Fit & Function replacement possible.
 - May be used by more then one users/clients
- The content of each Configuration Item must be defined
 - The order of increments must be clearly identified.
 - Each increment of a CI is developed in a separated configuration management branch
 - For each Configuration Item the start moment of development and the moment that the CI returns to the life-line is determined
 - The CI project life-line is used for synchronization across increments / branches.
 - Software changes between synchronization points on the CI live-line should be merged. These *synchronization moments* must be sequential in time in order to avoid merging problems in daily practice, and therefore must be planned in advance
 - Work in progress can be delivered from an increment branch, with the desired Quality level.
 - Final release will be performed from the CI_Project life-line, and can be a starting point for a next release.

Quality levels for a CI:

In the picture below quality levels are identified through which an CI can go through during incremental development. Normally the *release* level is only reached once during the project lifecycle, the *published* level is reached once every increment and the others can be reached several times during an increment:



8.2.7 Participants & collaborations



The following functions are crucial for this pattern:

- Architect: Defines the CI's, product content per increment and synchronization moments on the project life-line
- Project manager: Defines the number of increments together with the architect
- Test Manager: Defines the quality levels needed during every increment
- Build Manager: works out product archive structure and assures consistency across the increments CI's and product archive.

Architect, Project Manager and Test Manager must cooperate well to ensure all pieces of the puzzle fit well together: they must plan Incremental Configuration Management in advance and discuss impact of changes during the course of the project together.

8.2.8 Consequences

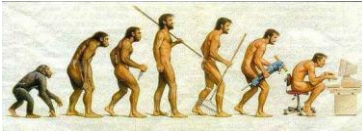
The main goal of this pattern is to enable that the product archive remains stable during incremental development. As a consequence, the project test team can always have a testable product at their disposal and therefore can ensure quality of the final product from the very first increment onwards. Also, if needed, the decision can be taken to release a certain increment, without having to wait for the next increments to finish. (eg. respond fast to market changes).

8.2.9 Implementation

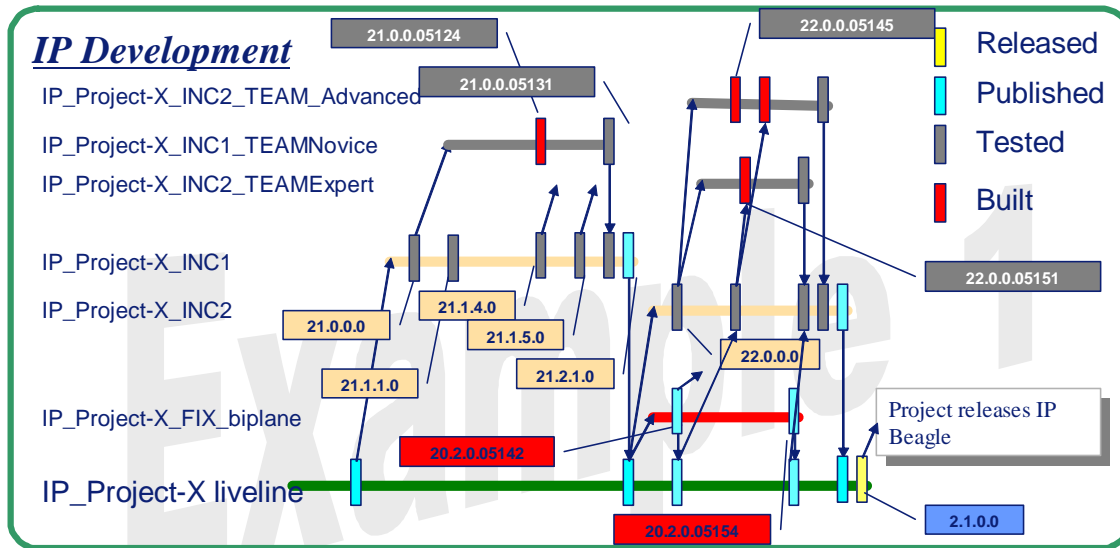
Aspects to be taken into account for this pattern are:

- Do not choose your configuration items too small; the more configuration items you get, the more complex the branching & merging of configuration items will get.
- Limit the number of nested branches per increment. Again here, the risk occurs of getting too much branches, which all have to be build by the build manager
- A configuration management tool must be chosen to implement this pattern. Any tool that supports branching and merging will do.

8.2.10 Sample



Example Incremental configuration management



8.2.11 Known uses

Philips Medical Systems

8.2.12 Related Patterns

Although many books are written on configuration management, no other configuration management pattern like above described has been found so far.

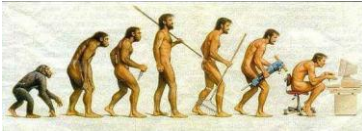
However, some of the items described in this pattern can be found in: High-level Best Practices in Software Configuration Management by Laura Wingerd & Christopher Seiwald from Performer Software

8.3 Defect Rootcause Analyses pattern

8.3.1 Pattern name and classification

Defect rootcause analysis

8.3.2 Intent



Defect root cause analysis is a process / method to enable organizations to determine the weaknesses in their development processes and products and to decide what changes they need to make and where they have to be introduced.

The intention is to encourage individual learning and to transfer from individual learning to organization learning about our mistakes.

This is done by identifying the root causes of all defects (problem reports (PR)), which are submitted during the development process of a product.

In this way we want to:

- Develop organizational understanding of the causes of a particular class of defects
- Fight the cause, not the symptom
- Learn from the mistake, not make the mistake over and over again
- Prevent defects, instead of solving them
- Shift reactive responses to defects toward proactive responses

8.3.3 Also known as

Other names or related methods are:

- Fault tree analysis
- Change analysis
- Causal Factor Tree analysis
- Fish-bone diagram or Ishikawa diagram
- Failure analysis

8.3.4 Motivation

If there is an unwanted situation, which consumes resources and tends to happen in a repeated fashion then it might be beneficial to figure out what is really causing this situation to occur and remove it so the situation does not occur again.

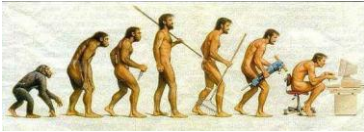
Benefits of this approach are:

- Reduction of defects
- Product quality improvement
- Increase of customer satisfaction
- Less time needed for testing and defect solving
- Shorter development / project duration time
- Decrease of development costs
- Increase of income

8.3.5 Applicability

The pattern is not only applicable throughout the complete product development lifecycle, but is also applicable to many other fields of origin:

- Safety based root cause analysis
Accident analysis and occupational safety and health
- Production based root cause analysis
Quality control for industrial manufacturing
- Process based root cause analysis
Analysis of business processes



- System based root cause analysis

8.3.6 Participants / Collaborations

The following functions are involved in this pattern:

- The developers, testers, designers, architects do the investigation and analysis of the root cause of the each individual defect and the information is saved in the defect record in the defect tracking system.
- Managers (project managers, resource managers, process owners etc) have to motivate and instruct people:
 - to perform the root cause analysis,
 - to deploy the process, to analyze the general root cause of a specific class of root causes (to define trends or patterns)
 - to define and implement improvement actions.

Management should also force a breakout of the pressure-driven reactive habits and use the accumulated knowledge to drive lasting improvements and to open the way to a proactive behavior.

8.3.7 Consequences

After analysis of the root causes of the defects one must define and implement the improvement or corrective actions. The actions might be:

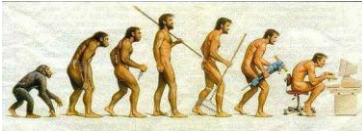
- Improve requirements management and system engineering process (i.e. content review, traceability)
- Introduce performance engineering (i.e. performance modeling, budgeting and measurements)
- Increase use of static & dynamic code analysis tools (i.e. coding standards checking, memory leak detection, code coverage analysis)
- Improve tests (i.e. test strategy, test environment, test automation)
- Extend training on architecture and application domain and improve system design skills
- Improve review process (i.e. train employees, act on metrics, deployment, management attention)
- etc

A precondition for doing defect root cause analysis is that the analysis and the administration of the findings about the root causes are done properly. (Uniform way and with the right quality level.)

This root cause analysis activity must be taken into account in the project and department planning as well as the deployment of the results and the associated actions (budget, time).

8.3.8 Implementation

Defect root cause analysis is a process designed for use in investigating and categorizing the root causes of defects with product development and production impacts (quality, reliability, robustness etc).



Simply stated, Defect root cause analysis is a method designed to help identify not only what and how a defect occurred, but also why it happened.

Only when investigators are able to determine why a defect has occurred, they will be able to specify corrective actions to prevent, that this type of defects will occur once again.

Understanding why a defect has occurred is the key to develop effective actions to solve the defect now and in the future.

The Defect root cause analysis is a four-step process:

1. Data collection
2. First order root cause analysis
Defect root cause classification and presentation of results
3. Second order root cause analysis
4. Action generation, implementation and monitoring results

The first two steps of this process is the Pareto of the defect root cause analysis.

8.3.8.1 Data collection

The first step in the analysis is to gather data. Each defect has to be analyzed deeply. Without complete information and an understanding of the defect, the causal factors and the root causes associated with the defect cannot be identified. The majority of time spent analyzing a defect is spent in gathering data. All the data is written down in the defect record in the defect tracking system. The defect tracking system has been extended with a root cause categorization field to classify the root causes of the defects in predefined classes or categories (where are the most root causes of the defects generated).

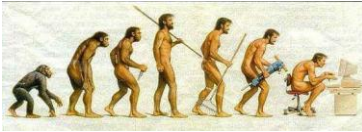
Examples of first order classes are:

- Requirements
- Design
- Implementation
- Installation / configuration
- Defect hardware
- Process
- Subcontractor component

8.3.8.2 First order root cause analysis

In this step a first filtering of the total number of defects is done using the first order root cause classes as defined above.

After a number of defects have been analysed and the root cause categories have been filled in correctly, as described above, one can make an overview of the root cause classes to find out what the major root cause classes are on which one has to focus on for a more detailed analysis (critical in the whole process is to focus on a reasonable subset of all defects). This is done with the help of the so called "Four-blocker" sheet. In this sheet an overview is shown of the total number of defects and an overview of for instance the four major root cause classes of the analyzed defects. (Pareto diagram of the defect root causes)



8.3.8.3 Second order root cause analysis

The second order root cause analysis is the next step in analysing the real root cause of a defect. In this step a more deeply analysis of a specific root cause class will be done. There are a number of dimensions that may in fact be at the root of each of the defects, that is, there may be several underlying causes rather than just one. The second order root causes can be divided also into a number of categories, for example:

- Phase related root causes
- Human related root causes
- Project related root causes
- Other root causes

Phase related root causes

These root causes are related to the standard development phases or documents: requirements, architecture, system design, unit / component desing, implementation etc.

Qualifications of phase related root causes are:

- Incorrect,
- Incomplete
- Ambiguous
- Not aligned with customer needs
- etc

Human related root causes

- Change coordination
- Lack of domain knowledge
- Lack of system knowledge
- Lack of tool knowledge
- Lack of process knowledge
- Individual mistake
- Introduced with other repair
- Communication problem
- Missing awareness of need of documentation or defined way of working
- etc

Project related root causes

- Time pressure
- Management mistake
- Caused by other product
- etc

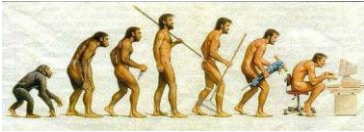
Other root causes

If needed other root causes can be defined as well of course.

Tools that can be used to identify the real root cause are:

- Fish-bone diagram or Ishikawa diagram
- Fault tree analysis
- Root cause map

Corrective action generation, implementation and monitoring results

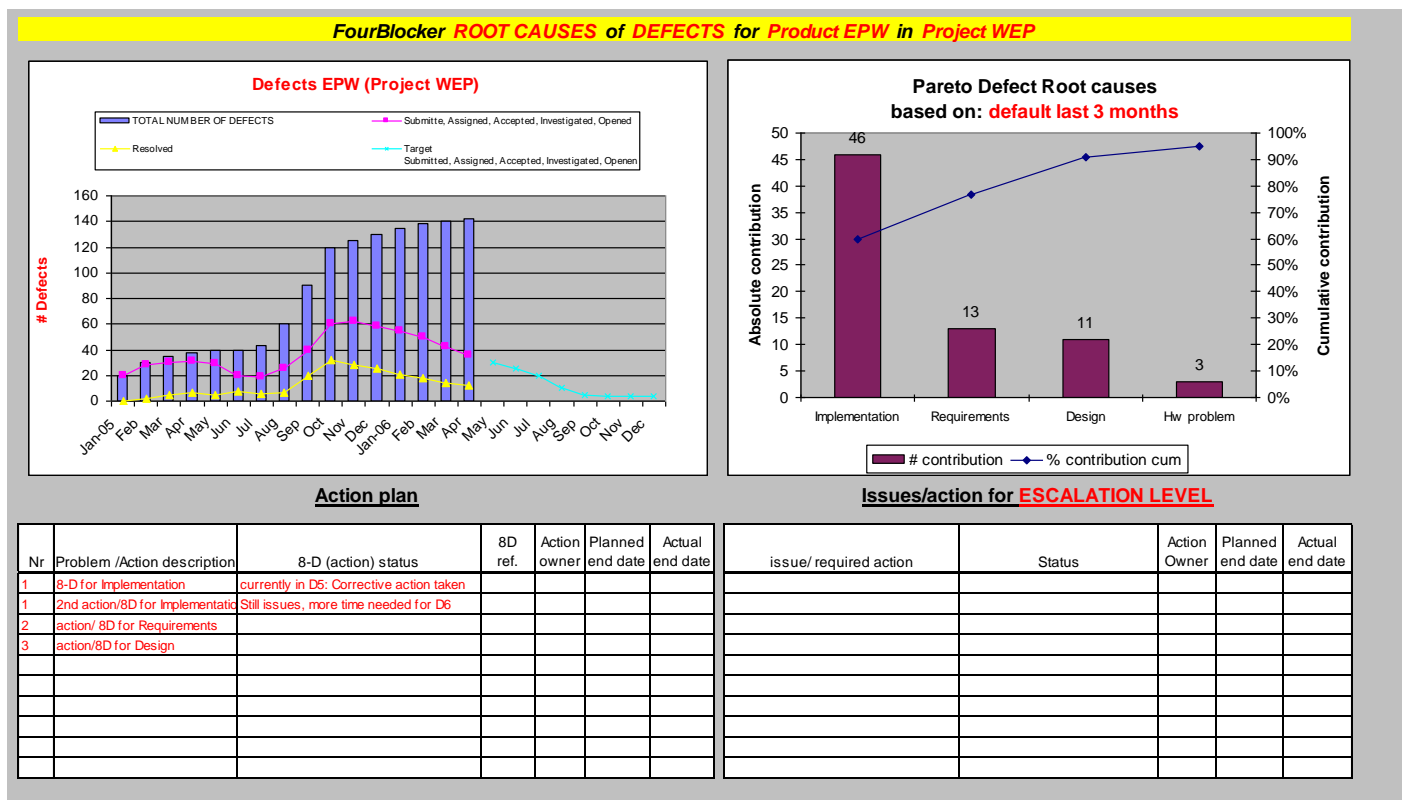


The last step is the generation of corrective actions for preventing the recurrence of the identified root causes of the defects.

The actions should directly address the root causes identified during the investigation. The rootcause analyst is often not responsible for the implementation of the corrective actions proceeded from the analysis. However, if the actions are not implemented, the effort spent in performing the analysis is wasted. Organizations need to ensure that the corrective actions are tracked to completion and results are monitored to show the effect of the actions.

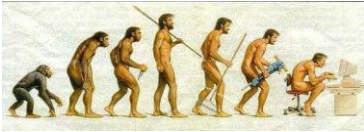
The corrective actions are also shown on the “Four-blocker” sheet.

8.3.9 Sample



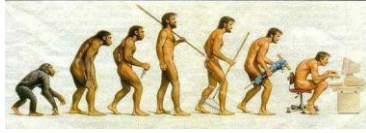
8.3.10 Known uses

This method is also used to find the root causes of the problems encountered in manufacturing the medical systems of PMSN and to startup improvement actions to reduce the manufacturing problems and to increase the product quality.



8.3.11 Related patterns

Incremental testing as used by PMSN. Defect root cause analysis on the encountered defects in the previous increment tests can be used to prevent defects in the next increments.



8.4 Proactive Quality Assurance pattern

8.4.1 Pattern name and classification

Pro-active Quality Assurance is a way to implement QA activities during a project or increment.

8.4.2 Intent

To provide project staff, management and stakeholders with appropriate visibility into the processes used within the projects and the deliverables created. Prevent non compliances and improve effectiveness of process execution by pro-active attitude, such as signaling, facilitating, monitoring, coaching and measuring and analyzes.

8.4.3 Also known as

Process Quality Assurance
Product Quality Assurance
Project Quality Assurance
Quality Management

8.4.4 Motivation

Normally QA is reporting at the end of an increment or project to control the right use of the processes and procedures. They are not involved in the daily way of working in the project.

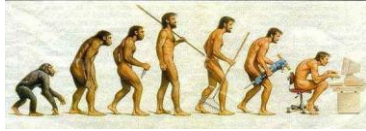
Results of the late reporting:

- Hard to recover or repair mentioned issues.
- Not taken seriously because timing priority.
- Seen as police agent.
- Problems in the processes are not been solved.

With pro-active quality assurance you are capable to inform your stakeholder about the quality aspects of the project at any time of the project, this in contrast to the normal QA-reporting after a release or increment closure.

Other motivation aspects:

- QA will be part a real participant of the project team.
- QA process and quality problems will be discussed early in time.
- At any given time, the Quality Officer can inform management clearly on the remaining quality risks.
- No quality surprises at the end of a project.
- Give the organization information to improve their Quality System and their way of working



8.4.5 Applicability

Pro-active QA can be applied for all product realization projects and all kind of development processes.

Typical situations are:

- Time-to-Market driven projects
- Technology driven projects with high uncertainties or that are difficult to plan
- Projects with complex software/hardware combinations
- Projects with a software only context
- Projects with very heavy quality requirements (medical or aviation)
- Complex organizations with complex quality systems.
- Organizations who have to deal with external norms and rules

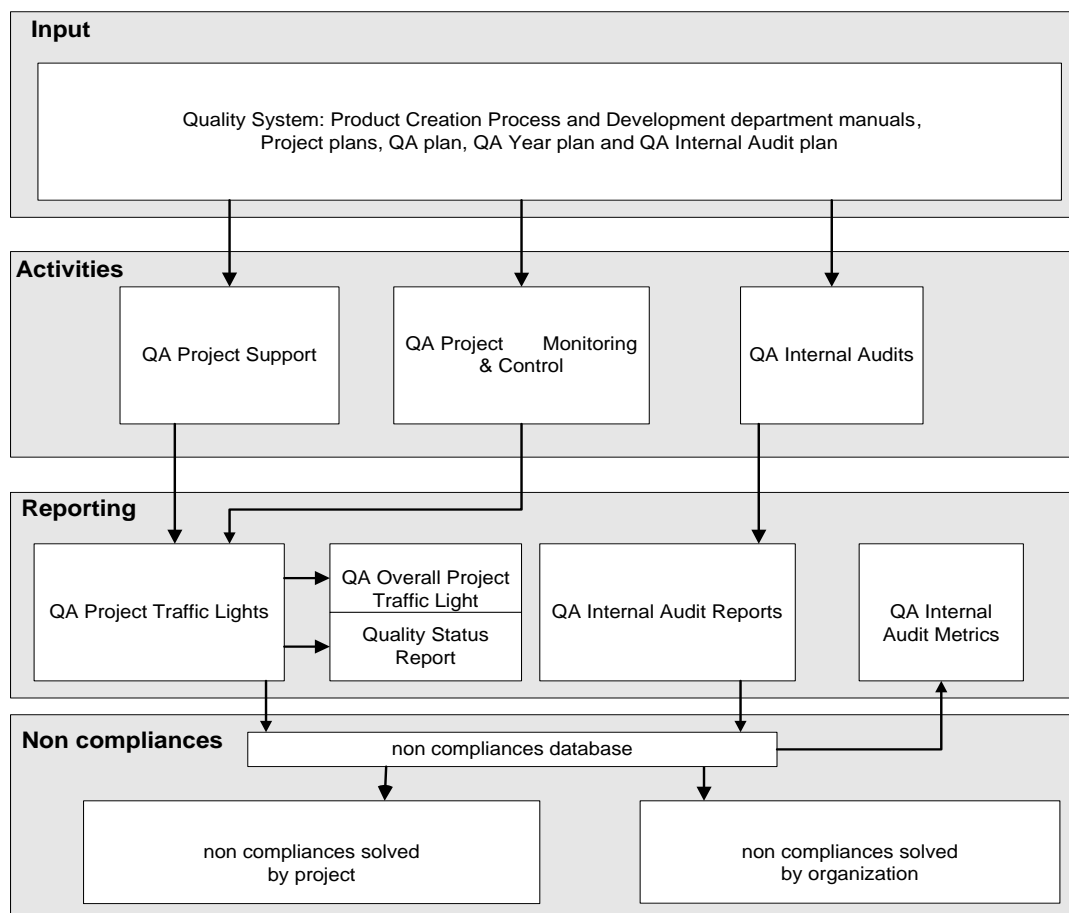
In general, difficult to plan projects with high risks in combination with Time-to-Market pressure.

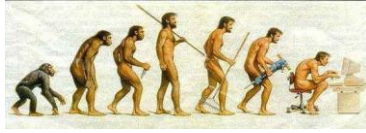
8.4.6 Structure

Pro-active QA has two main activities:

- Daily activities as project support, monitoring & control and reporting.
- Internal audits.

The figure below describes the areas of responsibilities of the QA Officer





8.4.7 Participants

At least the following roles participate in Pro-active QA Process:
Project Managers, Team Leaders, QA Management, QA Officers, Management Team of the organization, participants within the projects

Related processes are:

- All project management processes
- (Peer) review processes (walk through, inspections, comment collections and code reviews)
- Documentation processes
- Verification and validation processes
- Escalation processes

8.4.8 Collaborations

The base for pro-active QA is the collaboration between the project manager / team leaders and the QA Officer. They have to have contact frequently to talk about the plans in relation with the Quality processes and procedures in formal meetings as well as informal “coffee machine” talks (support and monitoring).

Further there has to be a frequent contact between the project participants to hear and see the daily way of working and where necessary to deploy the procedures or to train the participants (support, monitoring and control).

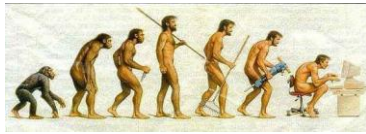
There is a less frequent (monthly) contact with the stakeholder to invest their satisfaction, possible wishes and improvements.

Organizationally there is a monthly feedback session with the line management to give the QA opinion about the status of the projects and about the status of the processes.

8.4.9 Consequences

This pattern supports and proves the objectives by:

- Facilitates the communication between all parties involved in the running project
- Clear reporting possible at any time in the project for all kinds of participants
- Early detection of possible quality issues
- More knowledge and understanding within the projects and the management of the actual quality processes and procedures
- Tracking of quality issues during the development (not afterwards).



8.4.10 Implementation

This way to implementing quality assurances ask a lot of the skills of the Quality Assurance Officer.

In special the QA Officer has to be:

- A net worker, working by walking around
- Confidently, straight, open-minded
- A motivator.

Other point of attention is the time you have to spend on quality issues early in the project, but this investment is proven less than the time and money you have to spend by finding quality issues late in the project.

Last main requirement is the separate communication line to the management of the organization. Necessary to improve issues over project level and to escalate issues between the project leading and the QA Officer within the project. Quality Assurance always needs a independent escalation line.

8.4.11 Sample


Example of front page of monthly traffic light report used for information to the project participants and input for the QA Overall project traffic light report :

Monthly DQA Traffic Light Report

XDY040- 070207

Date: 0704

Overall Process Status



Project: **Project X**

Project Leader(s): Pro Jectleader

lifecycle: System Design, phase 2

Increment:

Author: QA Officer

DQA Color Motivation

Status: Running to RFO. Delayed to week 0703

Subsystem FAT's are not started on time. Project has delay on delay on this moment.

Before start subsystem FAT there are more than 2800 PR/CR's assigned to the project X

Subsystems and system have problems with short and long term activities.

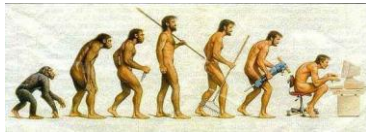
The projectplan is still not ready (see red in actions)

Project plan eagle is not in authorized state.

Lot of unsigned (> 20) documents in the archive.

	PM	SD	CM			Subpm	Test	Remarks
			Intergr.	CM	Test			
Project Management	Nok	Nok	OK	OK	OK	NOK	OK	Project plan still missing Project plan eagle is missing
Engineering	N/A	Nok						Realisation of FDS documents is not clear First steps to come to traceability are not visible yet
Support	Nok	Nok	Nok	Nok	Nok	Nok	NOK	Archive must get attention because it is not up to dat
Quality Targets	N/A	N/A	N/A	N/A	N/A	N/A		

Action Id	Description	Agree	Holder	Due da	Status
0543-04 (RB1)	Define (& deploy) estimation, planning, tracking & riskmanagement for SD--> CMMI	Yes	XX	0612	Forwarded
0617-01	Creaa a.s.a.p. a Overall project plan including a clear scope and clear specs	Yes	XX	0623 0635 0638 ESC	Open
0639-PPC	Make plan for PR-soving Tracking via PPT	Yes	XX	0640 0645	Closed
0639-PPC	Discuss outcome PPC with operations Jappie Dekker	Yes	XX	0640 0645	Closed
0639-PPC	What does backwards compatibility with "older" versions mean? Not necessary for XtraVision	Yes	XX	0640 0645	Closed
0639-PPC	Missing or not authorized deliveries: Security Risk Assessment, SRS , RAR, Application Report	Yes	XX	0640 0645	Closed



Quality Assurance driven Process Improvement. This pattern shows how to derive & implement improvements from non-compliances found by QA audits.

8.5.2 Intent

To prevent pro-actively future process non-compliances, which are currently found in several development projects. The approach is that the non-compliances are solved by removing the root cause that they will not be found in new development projects.

8.5.3 Also known as

- Process/Product/Project/Software/Development Quality Assurance
- Software/Development Process Improvement
- Quality Management
- Learning organization

8.5.4 Motivation

In development departments an independent Quality Assurance group has been institutionalized. This Quality Assurance group verifies periodically process compliance in projects and reports the results to project- and line management. After agreement with the QA representative the project manager is responsible to track the non-compliances to closure, meaning implementing the corrective action and removing the root cause.

When a development department executes several projects, one can observe that a number of non-compliances are only related to one project, but there will also be non-compliances which are visible in several projects. When a non-compliance is identified in only one project, this is probably a project issue, because other projects succeed in implementing the process on the right way. The project manager is responsible for the corrective action.

When non-compliances are identified in more projects, a corrective action can be necessary on department level. Possibly the related process does not fit the organization or the process is not well deployed. In many departments only corrective actions on project level are initiated resulting in the observation that the same non-compliance is found again in new projects, since the root cause of this non-compliance has not been removed.

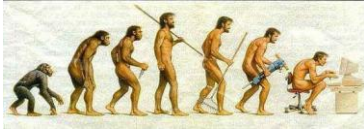
8.5.5 Applicability

This pattern can be implemented in all development departments running projects. Pre-conditions for implementing this pattern are:

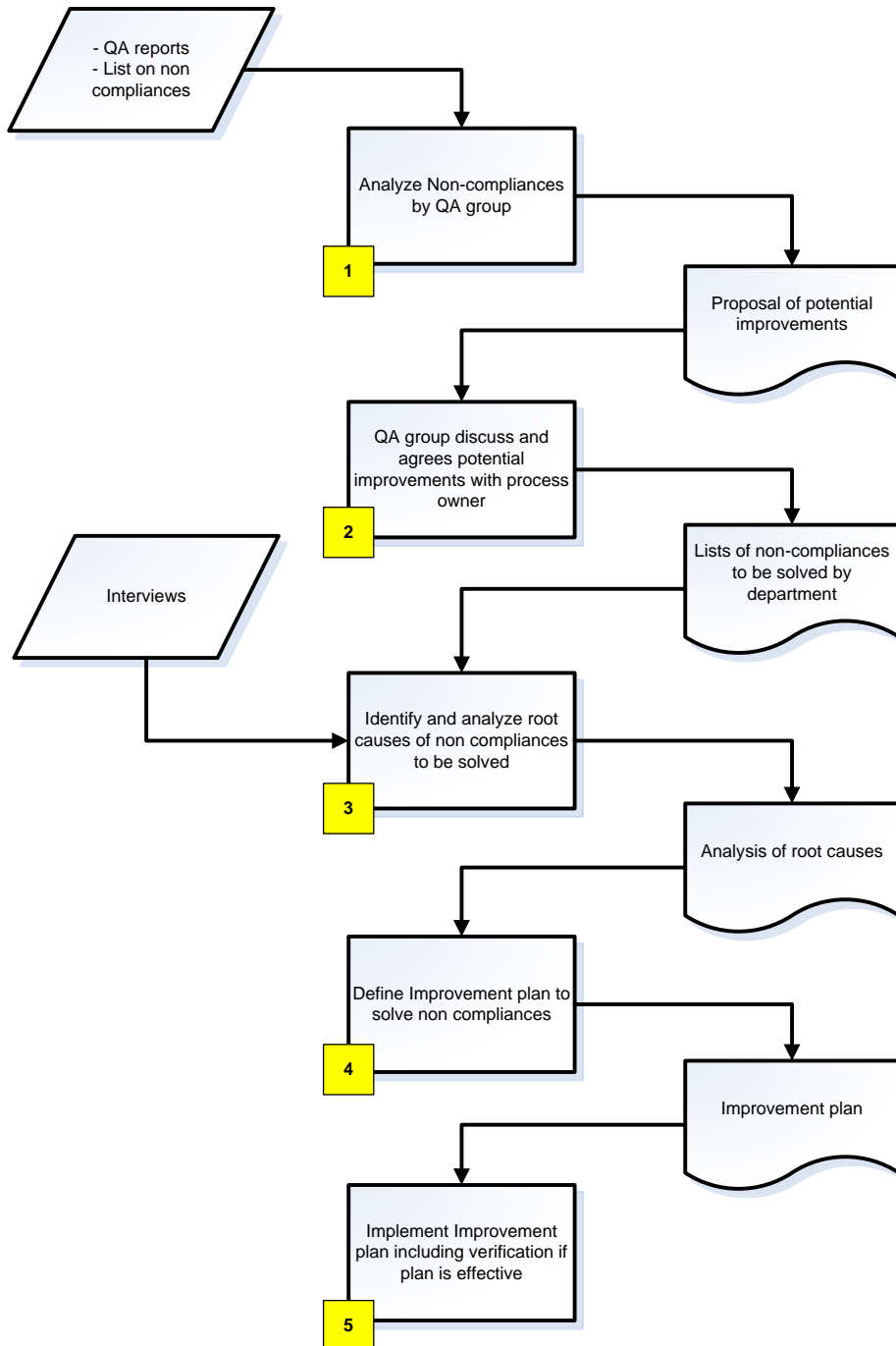
- A quality assurance group has been institutionalized. This group reports periodically the process status of the development projects and identifies the process non-compliances. These non-compliances can be traced to the related process area.
- Process (improvement) ownership is institutionalized. Examples of this ownership are SEPG, Process Improvement Coordinator, Ownership by development manager(s), ...

8.5.6 Structure

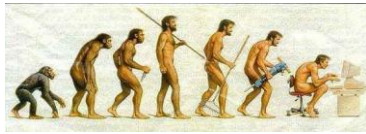
The process flow below shows the structure how Quality Assurance initiates process improvement activities in cooperation with the process owner.



1. First step in the flow is to analyze the list of non-compliances. Start with making a non-compliances list sorted on impact, followed by plotting each non-compliance in a matrix. One axis of the matrix defines the benefit level on impact (i.e. scale low till high) and the other the ease of implementation (scale easy till hard). Use this information to create the proposal list, start with "low hanging fruit" (easy to implement).



2. The Quality Assurance group discusses the proposal list of potential improvements with the process (improvement) owner and agrees with the non-compliances to be solved.



The Quality Assurance group starts with interviewing to identify the root-causes of the selected non-compliances and looks for best practices. During the interviews, the question “why” will be repeated till the real root cause is identified. It is also useful to find the rationale why other projects can produce best practices under the same circumstances.

3. When the root causes are known, the Quality Assurance group and the process owner define the improvement plan to solve the non-compliance. This improvement plan must comply with Plan Do Check Act (PDCA). The process owner is responsible for the implementation of the improvement plan. The Quality Assurance group checks the status of this plan and reports it to higher management.
4. When the plan is defined, one can start with the implementation. The implementation can only be closed after the verification shows that the corrective actions are effective.

This flow must be frequently repeated (e.g. quarterly).

8.5.7 Participants

Quality Assurance Group and Process (improvement) owner are participants of this pattern. They define the improvement plan and the project members implement it to ensure sustainable results.

All development processes are part of the product realization and be covered in this way and you can embedded this approach in the Quality Assurance process.

8.5.8 Collaborations

The base for this pattern is the collaboration between the Quality Assurance Group and the process (improvement) owner. They decide together about the content of the proposal list of potential improvements and ask project management to confirm this decision. Together they define the plan to solve the non-compliances. Project members will be involved during the implementation plan to get fast feedback and to accelerate the acceptance of the new way of working.

8.5.9 Consequences

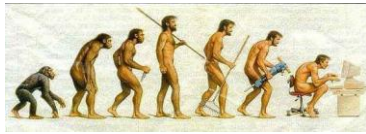
This pattern results that non-compliances which are identified in more projects are solved effectively. When non-compliances are not structurally solved on department level, the root-cause will not be removed and there is a big chance that the same non-compliance will return in new projects.

8.5.10 Implementation

The objectiveness and well argued Quality Assurance reports form the basis for this pattern. This improves the acceptance by the project manager and simplifies the identification of non-compliances.

Other point of attention is the root cause analysis of the non-compliance. When one stops too early with the question “why” during the investigation of a non-compliance, one discovers the symptom, but not the real root-cause. So don't stop questioning “why” until the real root cause has been identified

The final success factor of this pattern is that all corrective actions, including the check actions are performed thoroughly. For that reason it is recommendable to report the status of the improvement plan in the (development) management meeting.



8.5.11 Sample

Theoretical example for an overview of Quality Assurance reports listing non compliances.

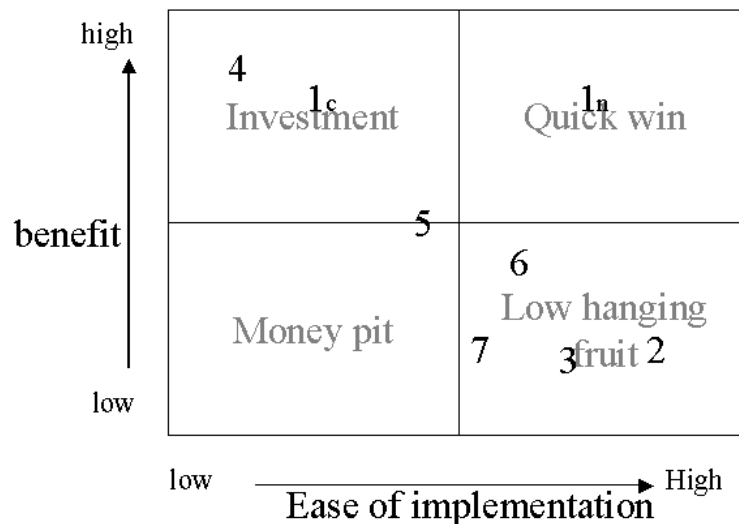
	Proj. 1	Proj. 2	Proj. 3	Proj. 4	Proj. 5	Proj. 6	Proj. 7	Process deviations in multiple projects
Requirements Definition & Verification								Requirement & Design backlog Requirement trading not done to all levels Missing requirements
Planning & Tracking								Open issues not planned Plans not up-to-date Size estimations and tracking
Configuration Management & Peer reviews								Review plan and tracking of reviews to closure in review tool. CM status reports on all disciplines EDHF checks shows that document control is not done properly
Supplier Agreement Management					n.v.t.			Discussion if SAM is applicable Project agreements not available
General								

Example of a matrix, used to select the potential improvements.

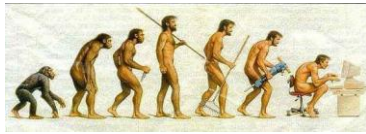
Proposal of potential improvements

For details on the fights, see Pre-selection of [fights](#).

1. Vertical traceability
2. Increment test plan
3. Review plan
4. Development Life Cycle – part x
5. CI promotion
6. CMP – what and where
7. Review tool



[c](#) indicates implementation within current projects, [n](#) indicates implementation in new projects.



An theoretical example of an improvement plan (PDCA compliant). The root causes are also recorded in this plan.

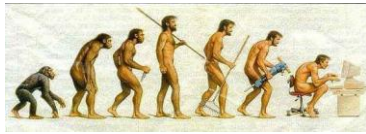
Philips Medical Systems		Last update		PROBLEM NUMBER	
8 DISCIPLINE CORRECTIVE ACTION REPORT		09-jul-2007, v07		PCP-07-07/Q2	
D1 STATE PROBLEM				TEAM	
Traceability not done in all projects from product requirements to lower level requirements and to design				<Name> PO	
				<Name> QA	
D2 PROBLEM DESCRIPTION					
-5 No/missing traces from Product Requirements to lower requirement/ design levels. -4 No/missing versions in document references -3 Decomposition mismatch (to unit/module names).					
D3 CONTAINMENT ACTION				status	
No containment action defined.				OK	
D4 ROOT CAUSE				AGREED	
				PO RD&VER <name>	
New: change management! 'Old' documents without any tagging are reused in new projects (n.a. for new products) Product requirements are too detailed (requirements which belong to lower level) or include design decisions. Difference between SMART and detailed is not understood. Unit/Module specs are created in parallel with the product requirements and design Requirements on Unit/Module level are specified by 'expert' people (autonomy). Related to this: no insight on effects on other units/modules.					
D5 CORRECTIVE ACTION		0.5		Action holder	Status
Promote best practice of traceability at other projects				Action holder	New
Output: deployment session (e.g. via powerpoint) in SFO. First promotion actions have been done					07xx
Make clear what the definition of unit and modules is (Focus on consistency architecture)				Action holder	Open
Output: deployment mail.					07xx
Define at early stage of the project the names of the units/modules.				Action holder	Closed
Output: presentation in department meeting					07xx
Promote naming conventions from SW to all disciplines, when possible.				Action holder	Open
Output: General guideline in Cbi					07xx
Reviewing with dedicated viewpoints: check on naming/ tagging consistency, ...				Action holder	Open
Output: Update of Review Guidelines.					07xx
D6 VERIFICATION OF ACTION		0.5		Action holder	Status
Verify the closure of the Corrective Actions defined at D5.				Action holder	New
Check if no new non compliaances are encountered for projects starting after wk7xx.				Action holder	Open
Check if problems (D2) are resolved in new (after wk727) projects.				Action holder	Closed
					07xx
D7 ACTION TO PREVENT RECURRENCE				Status	
Make new guidelines part of introduction training of new employees					
D8 CONGRATULATE TEAM					
Arrange diner for improvement team					

8.5.12 Known uses

Currently, this pattern is applied in Components X-ray, which is a Business Line within Philips Medical Systems.

8.5.13 Related patterns

This pattern can be implemented as extension on the pattern "Pro-active Quality Assurance".



8.6 Estimation in evolutionary SW development pattern

8.6.1 Pattern name and classification

This **estimation** pattern provides principles and practices that can be used during the execution of the process of project estimation. Although the contents of this pattern is somewhat focused on software estimates, the principles and practices also apply to estimates in other disciplines.

8.6.2 Intent

This pattern presents estimation methods/techniques, elaborates on size estimation and the use of historical data and gives a number of estimation examples. This document is intended for Project/Team Leaders and persons involved in estimates for projects or teams

8.6.3 Motivation and applicability

When estimates are made at the very beginning of a project, the level of detail is low and the uncertainties typically are at their largest. This is reflected in the accuracy (bandwidth) of the estimate. As time and the project progress, more detailed information comes available that can be used to refine the estimate. While the depth of decomposition of the system to be developed increases and uncertainties decrease, the associated bandwidth normally declines over time, as depicted in Figure 1

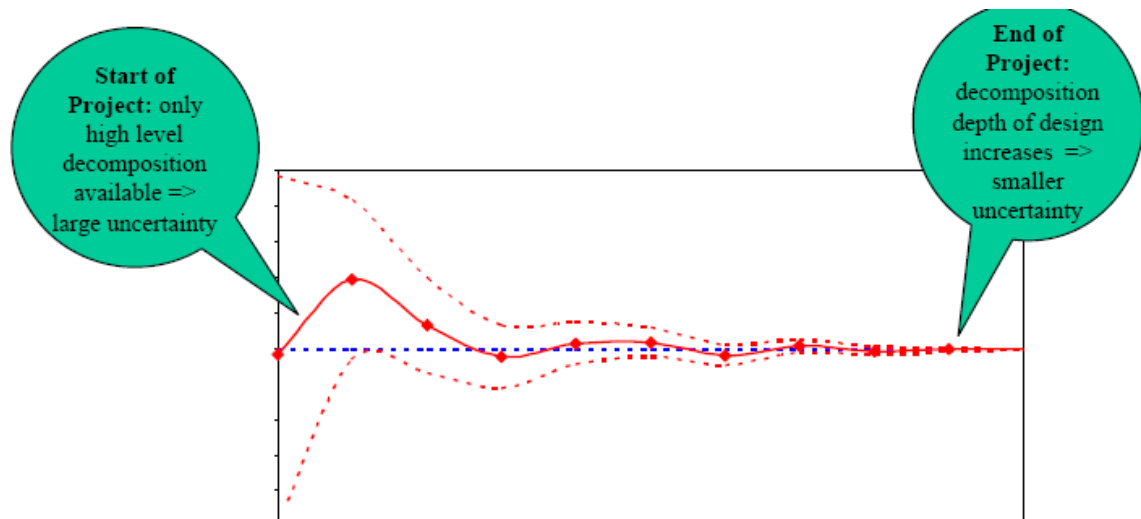
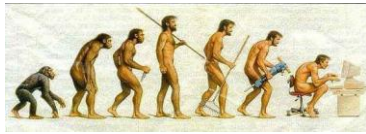


Figure 1 Estimation bandwidth

Initially, at the concept stage, a vague definition of the project may be available. Though the requirements may not be fully understood, the general purpose of the software to be developed can be recognized. At this point, estimates with an accuracy of plus or minus 50 percent are common.

After the requirements are reasonably well understood and, optionally, a feasibility study has been done, a function-oriented (e.g. requirement based) estimate may be



prepared. At that point typically around the scope commitment milestone, estimates with an accuracy of plus or minus 25 percent should be possible. Finally, after the project's (global) design phase has been done, an implementation-oriented estimate may be prepared. This estimate is based on the project's work breakdown structure (WBS) or on incoming Change Requests during the project's execution phase. This estimate is expected to be accurate within plus or minus 10 percent (see Figure 2).

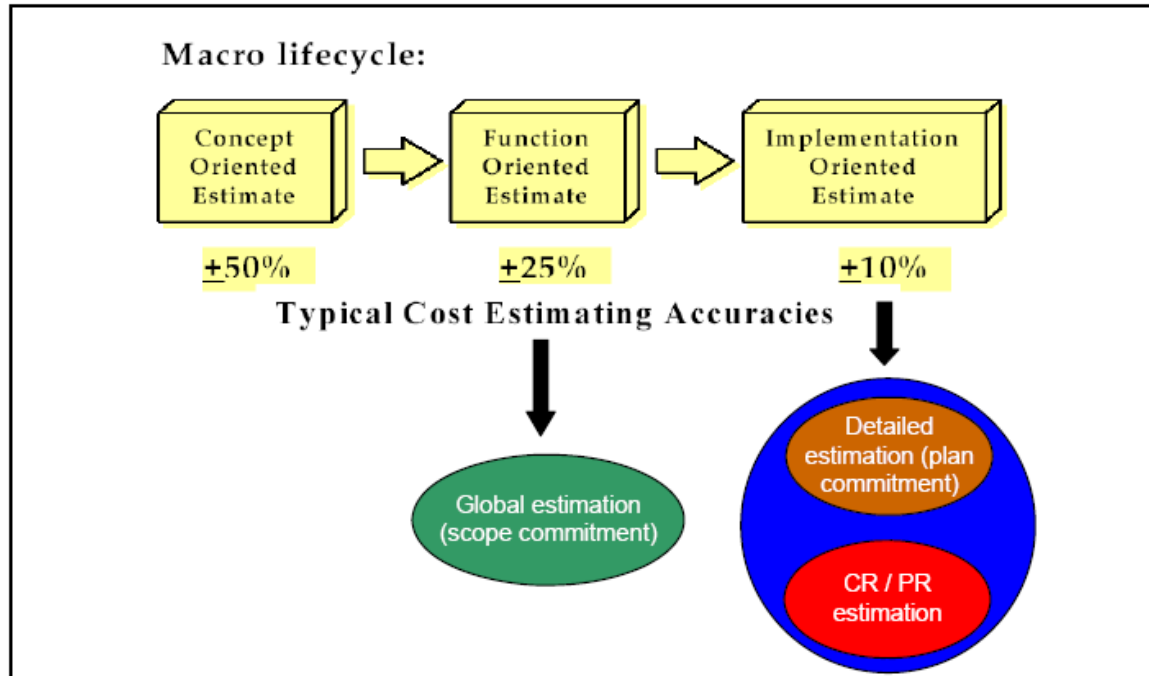


Figure 2 Typical estimating accuracies

It is important to recognize that estimation methods/techniques that can best be applied will depend on the (software) macro lifecycle stage. See Figure 3 for estimation technique selection criteria and related advantages and disadvantages.

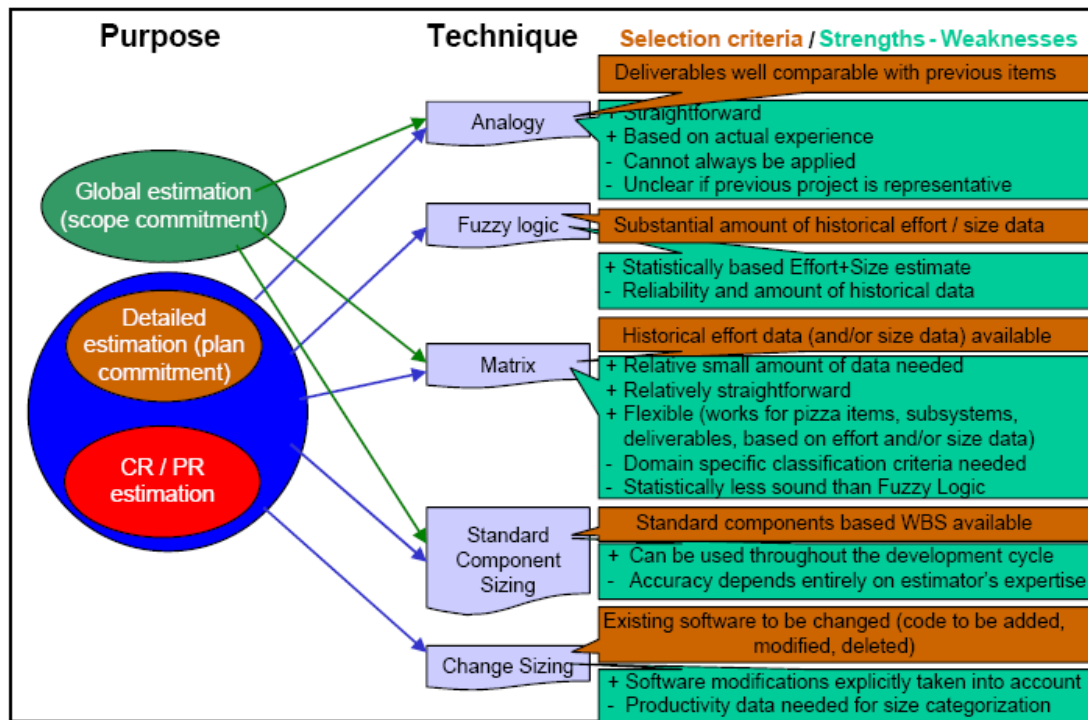
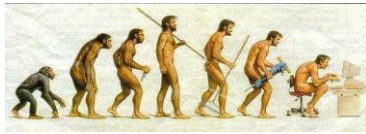


Figure 3 Estimation purpose versus techniques

Methods/techniques of estimation:

Analogy Method

This approach (see [BOEHM]) involves relating the proposed project to previously completed projects of similar application, environment and complexity.

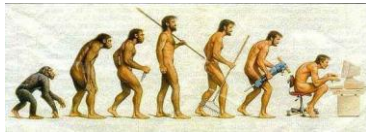
The basic steps are:

1. Break down the requirements to the granularity possible
2. Identify similarities and differences between previously developed work products
3. Identify comparable work products
4. Obtain the size information of the work products, using the data of the work product
5. Consider this size information as the basis for estimating the size of the current work product
6. Generate a size estimate
7. Calculate the required effort by multiplying the size estimate with the project's historical productivity figure

Fuzzy Logic

When no closely related work products are present but a number of work products in the same domain are available, it is possible to define a scale from measured sizes of the existing work products. Such a "fuzzy logic" scale has for instance five classes: very small, small, medium, large, and very large. The total size of the new work product is given by mapping it into one of these classes. The calculated standard size for the selected class is taken as the size estimate. The basic steps are:

1. Determine the size of all historical elements (e.g. code modules) in the same domain
2. Calculate the logarithms of these sizes
3. Calculate the average and sigma of the logarithms



4. Define the five typical class sizes as average + $n \cdot \sigma$, where n is -2, -1, 0, 1, 2
5. Calculate the reverse logarithm on the five values obtained
6. Categorize each element into one of the classes
7. Use the typical class size as the estimate for the element size.

Matrix Sizing Method

The matrix estimation method is an intermediate between analogy based estimation and fuzzy logic estimation. The basic steps are:

1. Identify a metric for size and a metric for complexity.
2. Setup a 3x3 matrix with Small, Medium, Large columns and Simple, Normal, Complex rows.
3. Find a number of typical historical examples for each of the nine boxes and enter their actual size and/or effort in the box.
4. For each box, determine the average size and/or effort.
5. Classify the item
6. Take the average value from the corresponding box as estimate (see Table 1).

	Small		Medium		Large		Total
	Count	Size	Count	Size	Count	Size	
Simple	3	8	7	64	4	256	1496
Normal	9	64	17	256	8	1024	13120
Complex	1	256	6	1024	2	4096	14592
Grand total							29208

Table 1 Size - Complexity matrix - Application example

Standard Component Sizing Method

Standard component sizing (see [SWEST]) is based upon information that is available with increasing precision from the feasibility study phase through the testing phase. This information is provided by designers, programmers, testers or others familiar with the project.

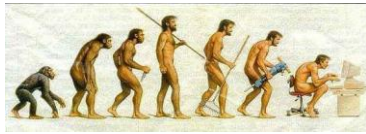
Change Sizing

This approach (see [SWEST]) estimates the size of a software system when it consists of new code and existing code that will be modified in various ways. The following categories of code can be distinguished: added, modified, deleted, and same.

The amount of new code can be estimated using any estimation technique. The amount of code in each of the other categories can be estimated using the standard component sizing technique. The principal merit of the change sizing technique is that it takes explicitly modifications into account. After having estimated the code size in each of the categories, effort can be calculated using effort ratios per category in combination with historical productivity data. In literature (see [SWEST]) numbers for the effort ratio for the code categories are available. As a starting set, the values in Table 2 are reasonable to start with.

Size Categories	Effort Ratio
Added	1
Modified	0.27
Deleted	0.15
Same	0.12

Table 2 Effort ratios



Wideband Delphi Like Technique (worked out in next chapters)

8.6.4 Structure

The Wideband Delphi like Technique is a supporting technique that may be used in combination with all other estimation techniques. Instead of several estimation rounds, a single estimation round is often applied for a Wide Band Delphi estimation session in this case. The Wideband Delphi method (see [BOEHM] and [SWEST]) described in this chapter can be used in several steps of the project estimation process.

This method is used to estimate the size of a work product or to estimate the effort required producing the work product. The Wideband Delphi method is a group approach for reaching convergence on estimating the size and/or effort of an activity. It is primary a systematic approach for sharing basis for estimation and assumptions amongst various people involved in estimating an activity and hence facilitate a more accurate estimation.

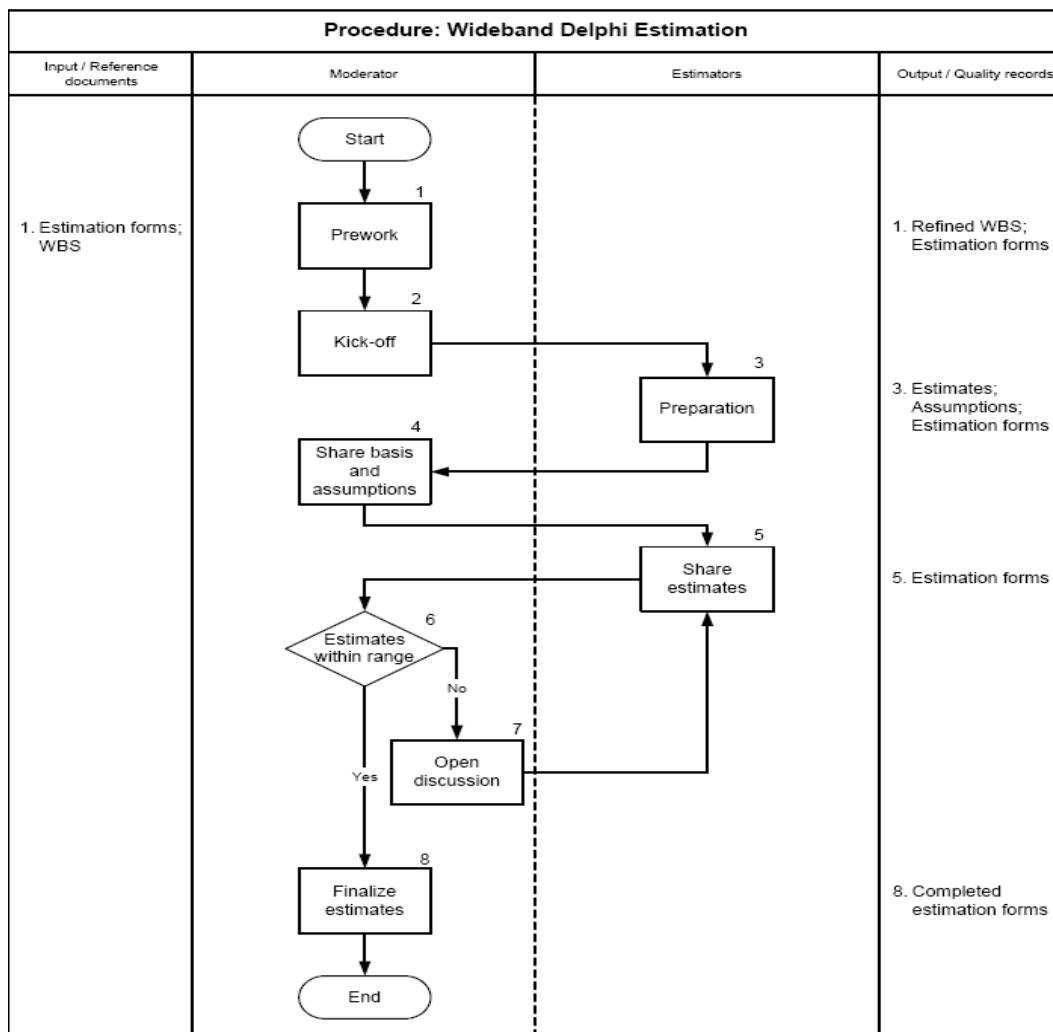
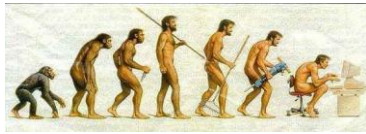


Figure 4 Flowchart Wideband Delphi like Estimation

Step 1. Pre-work

The moderator selects at least 3 estimators. The estimators preferably should have some stake in the activity. The moderator prepares the kick-off session by filling in the



estimation forms with all the default data filled in. Depending on the development cycle phase of the project, the architect will support him creating the list of deliverables and the corresponding work breakdown structure. Invitations are sent out to estimators with details of date, time and venue of the kick-off meeting. The Wideband Delphi Like estimation approach is explained for those who are not familiar with it.

Step 2. Kick-off

The moderator provides the estimators with all the input documents (including estimation forms with all the default data filled in), with details of the estimation session (date, time, venue, subject to be estimated (size3, effort and/or critical resource usage)) and estimation technique(s) to be used. The work breakdown structure is discussed (is it clear, is it complete) and general assumptions and cost drivers (e.g. complexity, available application experience) are looked at. Per type of work breakdown structure item a common view is determined and agreed. If necessary, the work breakdown structure is updated and re-distributed among the estimators. If needed, the estimation technique to be used is explained. The moderator sets the convergence range (e.g. 15 %), depending on the development cycle phase of the project.

Step 3. Preparation

Every estimator individually produces his own independent estimates of the size and/or effort involved for the activities. The estimates should be based on the estimation form provided; any extra or changed assumptions and newly identified risks should be recorded on the form.

Each estimator completes the appropriate columns in the table of the estimation form and the estimation session number and hands over his form to the moderator. The estimators should not share their estimates during preparation.

Step 4. Share basis and assumptions

The moderator checks that all estimators have completed the estimation forms, combines all estimates, assumptions and risks in one sheet and calculates the average estimates. The moderator invites all estimators to share their estimation basis and assumptions in the estimation session.

Step 5. Share estimates

The moderator invites all estimators to share their individual estimates.

The moderator collects the information forms from the estimators and completes his own estimation form with the averages, standard deviations and assumptions of the (re-)estimates.

Step 6. Estimates within range?

If the estimates for individual activities are all within the convergence range then the meeting is concluded (see step 8). Else, the moderator invites the estimators to re-estimate, based on the shared basis and assumptions (step 4) and to open discussion (step 7).

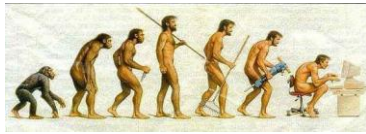
Step 7. Open discussion

If the deviation is greater than the convergence range:

- The moderator invites the estimators to discuss these points
- The moderator asks the estimators with dissenting opinion to argue their opinion.

Step 8. Finalize estimates

If the deviation does not reduce to under the convergence range, decide explicitly in a final discussion whether a dissenting opinion must be discarded as an outlier or that



the average of all estimates will be considered as the final estimate, consequently with a large variance. The amount of variance can influence the contingency to the identified risks.

A final estimate is agreed and recorded in the estimation form.

Exit Criteria (End)

Convergence or agreed non-convergence.

8.6.5 Participants

Project/Team Leaders and persons involved in estimates for projects or teams.

Roles:

	Responsibilities	Tasks
Moderator	facilitate the estimation session	<p>Appoint at least 3 estimators and optional additional estimators:</p> <ul style="list-style-type: none"> • Estimators are the developers who will (or are likely to) carry out the activities to be estimated. • Additional estimators include staff with previous experience of similar developments or with in-depth knowledge of the development being estimated. <p>Act as a chairman and ensure that the estimators do not get into unproductive discussions. Record the estimates. Conclude the estimation meeting with converged estimates or reconvene meeting at a later date in case of non-convergence</p>
Estimator	produce estimates	Produce estimates using a method the estimator is comfortable with

8.6.6 Collaborations

Collaboration between the several project teams, functional disciplines, team members and local experts is needed for good estimations.

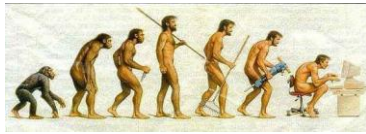
8.6.7 Consequences

Estimation reliability

Expert judgment based estimation depends heavily on the skills and objectivity of individual estimators and their past history developing similar products. Often when an expert-based estimation method is used, no consideration is made for the skills and capabilities of others to do the estimated task. A technique like Standard Component Sizing can help to minimize the impact of varying skills and capabilities on the estimate and can give an indication of our confidence in the estimate.

Historical data

Nearly all estimation techniques rely on the availability of historical data. Generally speaking, the more historical data is available, the better the estimations can be. So, it is essential that accurate and representative data from completed projects is available. The use of following historical project information can significantly contribute to better size and effort estimates:



- (Changed) size of project deliverables (e.g. number of new and modified Lines of Code (LOC))
- Effort expended on pizza items, project deliverables and project activities
- CR/PR information (e.g. counts, effort expended, severities)
- Review data (e.g. review effort, number of major remarks)
- Test data (e.g. test effort, number of planned and executed test cases)

Estimation Tips:

Allow time for the estimate and plan it.

- Rushed estimates are inaccurate estimates. Take the time to plan the estimation activity itself so that it can be done well.

Use data from previous projects.

- By far the most common practice used for estimation is comparison with similar, past projects based solely on personal memory. This practice is associated with cost and schedule overruns. The use of documented data from similar past projects will significantly contribute to cost and schedule precision.

Use developer-based estimates.

- Estimates prepared by people other than the developers who will do the work are less accurate than estimates prepared by the developers who will do the work. When estimator-developers do the estimate and the work, meeting their own estimates reflects positively on both their estimating and work abilities.

Estimate by walk-through.

- Have each team member estimate pieces of the project individually and then have a walk-through meeting to compare estimates. Discuss differences in the estimates enough to understand the sources of differences. Work until you reach consensus on the high and low ends of estimation ranges.

Estimate at a high level of detail.

- Base the estimate on a detailed examination of project activities. In general, the more detailed your examination is, the more accurate your estimate will be.

Don't omit common tasks.

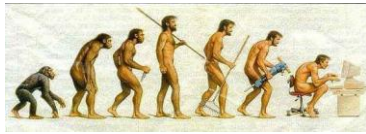
- People don't often omit tasks on purpose, but when they've been ordered to develop a product in the shortest possible time, they don't go out of their way to look for extra tasks.

Re-estimate in a later stage, when more information is available.

- When estimates are made at the very beginning of a project, the level of detail is low and the uncertainties and risks are typically at their largest. The more details are known about e.g. requirements, architecture, people, methods and tools, the more accurate the estimates will be. Therefore it is important to re-estimate as the project progresses.

Use several different estimation techniques and compare the results.

- Try several estimation techniques in order to avoid the weakness of any single method and to capitalize on their joint strengths. Study the results from the different techniques.



8.6.8 Implementation

Selecting size metrics

It is generally agreed that the size of the products is the predominant characteristic in determining how much effort is needed to build it. Suitable size metrics are those project attributes that, in the mind and heart of the estimator, lead to higher/lower levels of effort or longer/shorter project schedules.

Furthermore, good size metrics for estimation purposes are those that:

- Have a positive correlation with development effort
- Have historical data available
- Are objectively (and preferably automatically) measurable

"Objectively measurable" means that repeated measurements of the size result in the same value for the metric. Automation of the size measurement is implied and strongly recommended.

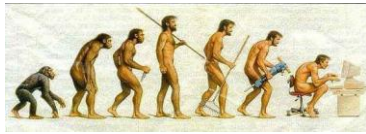
Relationship between size and effort can be derived when relevant and sufficient historical data is available. This means that comparable components exist from which the size-effort relation can be determined. Each type of deliverable or activity has a need for its own size definitions. It is important to find and use the right one each time. For examples see Table 2.

Deliverable/Activity type	Description of size measurement
Make/Review/Rework requirement specification	Number of requirements; Number of added + modified pages; Complexity/size categories
Make/Review/Rework design specifications	Number of classes; Number of screens (GUI); Number of added + modified pages; Complexity/size categories
Make/Review/Rework Code	Number of added + modified non-commented lines of code; number of classes; Complexity/size categories
Make/Review/Rework verification specifications	Number of added + modified test cases; Number of added + modified pages; Complexity/size categories
Make/Review/Rework verification reports	Number of test cases to be executed
PR solving	Number of PRs
Integration Test	Number of test cases to be executed

Table 2 Examples of size metrics

8.6.9 Sample

Example of an estimation sheet:



Estimation Sheet																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
Project:	<Project Name>				Use the Remarks column to enter your reasons, considerations, computation, analogy with, assumptions, reuse from, remarks and other information that led you to your estimates.																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
Estimation issue:	<Estimation Issue>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
Session date:	<Session date>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
XDB number:	XDB044- <i><synonym></i>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
Input Documents:	<input doc>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
Historical Data:	<historical Data>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
Estimation Round:	1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
Estimator:	<name>				Hours																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
Estimator initials:	Est. 1																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
Time Spend:																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
Date:	<date>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
Legenda: Size (Documents) = Created or modified pages Size (Code) = Non Commented Lines of Code (NCLOC) Size (Verification Specification) = Number of testcases Size (Verification Report) = Number of testcases to run Effort = Time needed for task in hours All estimates are excluding reviews and rework!																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
<table border="1"> <thead> <tr> <th colspan="2">(Preliminary) WBS component / deliverable</th> <th>Unit of Size</th> <th>Lowest</th> <th>Most likely</th> <th>Highest</th> <th>PERT Size</th> <th>PERT StdDev</th> <th>95% Size</th> <th>Remarks</th> <th>Lowest</th> <th>Most likely</th> <th>Highest</th> <th>PERT Effort</th> <th>PERT StdDev</th> <th>95% Effort</th> <th>Remarks</th> </tr> </thead> <tbody> <tr><td>1</td><td>Example: COPACK</td><td>Package Requirement Specification</td><td>#pages</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>2</td><td></td><td>Package Design Specification</td><td>#pages</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>3</td><td></td><td>Package Verification Specification</td><td>#TC</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>4</td><td></td><td>Coding & Testing</td><td>NCLOC</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>5</td><td></td><td>Package Verification Report</td><td>#TC</td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>6</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>7</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>8</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>9</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>10</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>11</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>12</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>13</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>14</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>15</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>16</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>17</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>18</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>19</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>20</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>21</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>22</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>23</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>24</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>25</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>26</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>27</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>28</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>29</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>30</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>31</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> <tr><td>32</td><td></td><td></td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> </tbody> </table>															(Preliminary) WBS component / deliverable		Unit of Size	Lowest	Most likely	Highest	PERT Size	PERT StdDev	95% Size	Remarks	Lowest	Most likely	Highest	PERT Effort	PERT StdDev	95% Effort	Remarks	1	Example: COPACK	Package Requirement Specification	#pages				0	0					0	0			2		Package Design Specification	#pages				0	0					0	0			3		Package Verification Specification	#TC				0	0					0	0			4		Coding & Testing	NCLOC				0	0					0	0			5		Package Verification Report	#TC				0	0					0	0			6							0	0					0	0			7							0	0					0	0			8							0	0					0	0			9							0	0					0	0			10							0	0					0	0			11							0	0					0	0			12							0	0					0	0			13							0	0					0	0			14							0	0					0	0			15							0	0					0	0			16							0	0					0	0			17							0	0					0	0			18							0	0					0	0			19							0	0					0	0			20							0	0					0	0			21							0	0					0	0			22							0	0					0	0			23							0	0					0	0			24							0	0					0	0			25							0	0					0	0			26							0	0					0	0			27							0	0					0	0			28							0	0					0	0			29							0	0					0	0			30							0	0					0	0			31							0	0					0	0			32							0	0					0	0		
(Preliminary) WBS component / deliverable		Unit of Size	Lowest	Most likely	Highest	PERT Size	PERT StdDev	95% Size	Remarks	Lowest	Most likely	Highest	PERT Effort	PERT StdDev	95% Effort	Remarks																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
1	Example: COPACK	Package Requirement Specification	#pages				0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
2		Package Design Specification	#pages				0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
3		Package Verification Specification	#TC				0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
4		Coding & Testing	NCLOC				0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
5		Package Verification Report	#TC				0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
6							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
7							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
8							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
9							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
10							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
11							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
12							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
13							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
14							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
15							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
16							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
17							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
18							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
19							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
20							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
21							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
22							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
23							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
24							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
25							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
26							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
27							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
28							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
29							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
30							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
31							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
32							0	0					0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	

8.6.10 Known uses

Currently, this pattern is applied in Philips Medical Systems.

8.6.11 Related patterns

All Patterns used in evolutionary development.

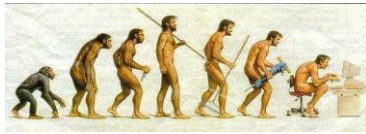
8.7 Baseline auditing and configuration status accounting pattern

8.7.1 Pattern Name and Classification

Configuration status accounting: reporting on the indicators that contribute to the status of a configuration item. This can be done continuous (e.g. daily) or on regularly (e.g. every 4 weeks).

Baseline auditing: check if those indicators meet the criteria for promotion to a new status. This new status typically is a requirement for passing milestones during a project.

Both are measuring the same indicators. The status accounting is way to check regularly if the indicators will meet the criteria for the baseline audit. This creates the possibility to start corrective actions in time when necessary



8.7.2 Intent

By performing Configuration Status Accounting on regular base, a prediction can be done during the project whether the criteria for the next milestones will be met or not. It gives project management the possibility to start corrective actions in time, so the final check (the baseline audit) will reveal no surprises.

8.7.3 Also Known As

Configuration auditing
Status accounting

8.7.4 Motivation

In a project, there are several milestones to be passed. In the days before passing the milestone, some people start running around very stressed to gather all kind of information to determine the status of the deliverables.

Examples of the information they're looking for are:

- which functionality has been delivered
- what is the status of our documents
- did everybody use the same interfaces and compiler
- has everything we promised been delivered

The stress becomes even more in case some of the answers cannot be answered or the answers are not satisfying.

But why wait until the days before the planned milestone to collect the information? By collecting and checking this information during the project, a prediction can be done on forehand whether the milestone can be passed or not at the planned date. Corrective actions can be defined in time, so there is much less stress when the milestone has to be passed.

8.7.5 Applicability

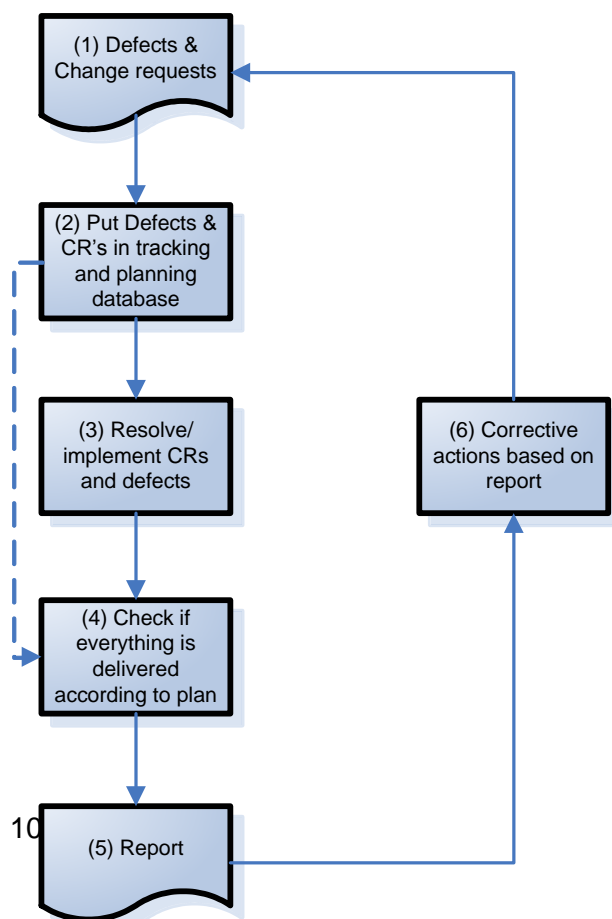
Depending on the Configuration Items checked with auditing and accounting, this pattern can be applied during several phases of the product lifecycle.

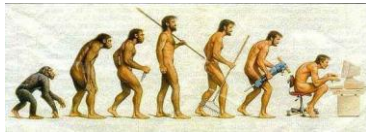
- During the scoping phase, the focus will be on documentation
- During the development phase, the focus will be on the functionality (code) and consistency
- During the maintenance phase, the focus will be on the defects (code).

8.7.6 Structure

Globally, the pattern can be drawn as shown.

Explanation of the steps:





- (1) Defects and change requests are the reasons to change the code/documents
- (2) The defects and Change requests are put into a planning and tracking database. Via this database the work is assigned to developers for execution.
- (3) Developers resolve the defects and implement the change requests. This will result in deliverables and a tracking record containing the applied change request/defects
- (4) This is the actual status accounting or auditing: comparing the planned defects/change requests against the delivered ones.
- (5) A report is written containing the findings and/or deviations from the planning.
- (6) Based on the agreements made in the CMP (Configuration Management Plan) corrective actions can be started to resolve the deviations or findings

8.7.7 Participants

Following roles are involved:

- Configuration manager
The Configuration manager is the one who actually performs the audit/accounting
- The role responsible for the deliverables (can be project manager, but also team leader or integration manager)
This role is responsible for having the deviations resolved / findings followed up
- Quality officer
The Quality officer has to guide the process and does track the follow up of the deviations / findings

8.7.8 Collaborations

The collaboration can be split over 3 phases:

- (1) Preparation: all people involved have to get the same “view” on the status of change requests, defects and planning. During this phase, also agreements are made on what is checked during the audits/accounting.
These agreements are usually written down in the CMP.
- (2) Investigation of deviations/findings: in case deviations and/or findings are found, these have to be investigated. It might be necessary that the involved people help each other in this
- (3) Follow up: the people involved have to agree upon follow up of the deviations and findings.
- (4)

8.7.9 Consequences

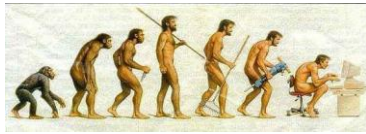
As mentioned before, the objective of this pattern is that corrective actions already can be initiated during the project, instead of at the end of the project when official deliveries have to be done.

Executing status accounting on regular bases (for example monthly) gives management an indicator of how the project (from CM perspective) is performing. When corrective actions are initiated (and executed!) in time, this pattern will contribute to a smoother passing of the milestones during a project.

8.7.10 Implementation

The biggest pitfall is that status accounting / baseline auditing leads to “management satisfaction”. This phenomenon is seen when the results of accounting/auditing are treated as a goal on its own, instead of supporting the project.

Some examples of “management satisfaction”:



- Team leaders adjusting the planning of change requests/defects just before the actual delivery, so they can deliver exactly what is planned. If they don't adjust the planning last minute, they will have a remark in the accounting report (so management attention). To prevent that, they adjust the planning.
- Manipulating data #1. People might keep an own "local" administration of defects found instead of putting them all in the defect database. Only one (general) defect is administrated officially, so to management it looks like they have very good software, but appearances are deceptive.
- Manipulating data #2. Sometimes a set of defects is made duplicate of a general defect like "improve this code". By creating these duplicates, the amount of open defects can be reduced very easy.

8.7.11 Samples Known Uses

Below 4 examples of how configuration status accounting / baseline auditing is used in existing projects.

1. Analyzing build results

Let's start with an example an analysis that is performed by most of the Software Configuration Managers (SCM) and/or build managers: the analysis of build results.

Analyzing and logging of build-errors on subsystem- or module-level might give insight into the quality of deliveries.

In case a certain module causes build-errors regularly, this might indicate gaps in the development process for this module.

Possible causes can be:

- Bad or no code review
- No functional test by the developer before delivery
- No build by the developer before delivery
- Developer doesn't know which versions of interfaces to use

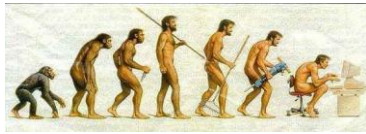
By monitoring the build results during the project, corrective actions can be initiated before it's too late.

A more in-depth investigation of the build-logging usually reveals additional information that might impact the project. Especially information with respect to usage of old (deprecated) interfaces which will be removed in a next release, or information about specific methods or functions that are not supported anymore in a next release of the compiler have to be monitored. This helps in preventing unexpected problems when an update takes place on the interfaces or compiler.

Typically, in the SCMP agreements on this are written down, for example that no old interfaces shall be used at a certain milestone. During status accounting it is checked if it's still feasible to meet this agreement and/or what's needed to meet it.

When the baseline audit shows that the agreements are not met, an impact analyses is necessary to find out if it will be a blocking issue for the milestone pass.

2. Maturity grid



Another measurement that suits status accounting and baseline auditing very well is the so called maturity grid (see frame). In the SCMP the maturity levels per milestone are defined. This makes checking of the maturity level a typical baseline auditing activity.

But, why wait until the last days before the milestone to check this? By checking the maturity on regular base during the project (as part of status accounting), a prediction can be done whether the required maturity level can be met in time or not. When needed, also corrective actions can be initiated.

If, for example, still 100 defects have to be solved to reach the required maturity level, but the milestone is within 2 weeks, the project has to see if solving 50 defects per week is realistic or not.

At such a moment, recording historical data pays off too. Based on this historical data, it's easy to determine whether solving 50 defects per week is realistic or not.

Maturity grid: a method where the maturity of the project is measured based on the amount defects raised, their severity and their actual status.

Representation is done by means of a grid. (S1-S2-S3 represents the severity)

	S1	S2	S3
New			
In progress			
Solved			

Per project phase it is defined which cells of the grid must have value 0 (representing the maturity).

Example: at the start of a test phase, all S1 defects have to be solved, and all S2 defects have to be in progress. So the grid has to look like below to start:

	S1	S2	S3
New	0	0	≥0
In progress	0	≥0	≥0
Solved	≥0	≥0	≥0

In case these criterion are not met, the amount of problems to be solved can be considered as an indication for the maturity of the product.

Sometimes it happens that people within a project maintain their own overviews of defects, or even personal maturity grids. The main objection of these personalized overviews is that everybody has a different view on the current status. Even when using the same queries on the defect database, the results might differ when one overview was generated in the morning, and the other one in the afternoon. This might lead to unnecessary discussions and/or misunderstanding.

This can be avoided by having overviews like the maturity grid generated centrally (e.g. by SCM) so all involved people talk about the same figures.

3. What did we actually deliver?

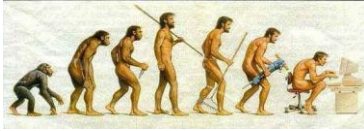
New deliveries in a software development project often leads to the question "what exactly does this delivery contain?"

Collecting delivery information used to happen by walking around and asking the developers what they actually delivered. Nowadays, this information often can be generated by coupling a SCM system and a CR/PR management tool.

But, why still discussions on what has been delivered? Most of the times, this is caused by different perceptions about the term "delivered".

A SCM engineer looks into his SCM system, and maps the changed configuration items to the related change requests and defects. This mapping results in an overview of what is delivered, and this is also what will be identified by means of a baseline. From SCM perspective, this overview is "the truth"

Others in the project, for example the team leader or integration manager, often look to the status of the change requests or defects in the CR/PR management tool and draw conclusions based on this information.



In a perfect world, these 2 overviews are identical. However, in the real world they often differ.

An example of how this can happen is a developer, who quickly has to fix a problem in his local environment just before he goes on holiday. The solution is delivered as a patch directly to the testers, or even worse, directly to the customer (without involving configuration management). The developer finishes the administrative part of his work (e.g. setting the defect status to solved) so from the perspective of the team leader everything seems o.k. now for this problem, since the status is solved.

But, as the developer is eager to go on holiday, he forgets to deliver the solution to SCM. As a result of this, the problem re-occurs after the next official delivery, so the customer will not be happy.

By comparing frequently the administration with the SCM environment (status accounting) problems in a later stage of the project can be prevented. At the milestone pass (the audit) there should be no more (unexplainable) differences between the two systems.

4. Consistency

Consistency is a typical keyword in the definitions within the scope of SCM. But, what is actually meant with this term, and how can it be checked?

One of the meanings of consistency is “having no inner contradictions”.

Assuming a software project has at least 2 types of configuration items (being documentation and code), three flavors of consistency can be defined:

Document – document

With a consistent set of documents is meant that the complete document structure is a consistent entity. In other words: all mutual references, including versions, are correct. After all you want to be sure that for example design specifications are based on the correct version of the system specifications. A solid and reliable document management system can help you in keeping the document structure consistent, but checking is still very often a manual task.

This checking can be automated too, but this really depends on using strict templates and conventions for documentation

Document – code

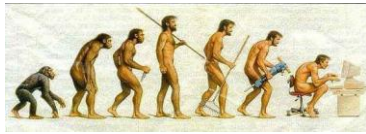
In an ideal project, stable requirements, together with a requirement management process and code reviews that the final delivery contains exactly what has been requested.

But....what in case one of these processes contains gaps?

This partly can be resolved by adjusting the SCM procedures and tools, so the gaps are closed. However, by solving it this way, SCM becomes owner of problems which actually are no SCM problems but organizational problems.

If, for example, there is no or inadequate requirement management tooling, SCM can support by creating procedures in the SCM area. On one hand the problem is (partly) solved this way, on the other hand, the root cause (inadequate tooling) is not tackled.

The role of SCM should be restricted to a monitoring one in this. Very detailed checks are very time-consuming, and only should be done on at random.



Usually, these random checks can be used as an indicator whether the relations between code and documentation is ok.

There are also some simple checks to think of, which are easy to implement. Assume a change is applied on a requirements document; then a logical continuation of this change would be a change on the underlying design spec and next in sequence the source code. If this logical sequence cannot be detected, this might indicate a gap in the processes.

Code-Code

During software development projects it's also important to pay attention to the consistency of the delivered code.

Focus should be that every part of the development process (developing, building, testing) has been executed in the same environment. Consider in this context aspects like compiler versions, libraries, standards but also interfaces between subsets of the product.

This kind of checking is especially applicable if the end product is assembled based on subsets of several (third) parties. But also in a local environment it's important that all involved are using the same versions of software and interfaces.

An example of what might go wrong in case the development- and build-environment are inconsistent:

The scenario is software development using JAVA. By default, JAVA version X is installed on all build- and development engines. At one day, one of the developers found a nice website, but to be able to view this site correctly, he needed a newer version of JAVA. So he downloaded this version and installed it locally on his development-PC. After he finished viewing the website, he continued developing code, but now (unintentionally) based on this new JAVA version. As this version had some nice new features, he made use of it in his code.

After the local tests on his own machine passed successful, he delivered the code to SCM. But, SCM couldn't get the code compiled. It took a few days to figure out that this was caused by the mismatch in JAVA version.

8.7.12 Related Patterns

All other configuration management related patterns

8.8 Software Development Stream pattern

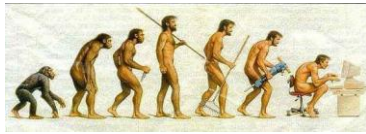
8.8.1 Pattern Name and Classification

Software Development Stream pattern.

8.8.2 Intent

This pattern describes a branching approach for software development streams. The context wherein it is applied is:

- Multiple development projects running in parallel
- Evolutionary development with backwards compatibility requirements



- Multi-site development
- Development of an embedded system, which has to deal with both a hardware and a software development lifecycle.

8.8.3 Motivation

A generic overview of branching strategies is described by Brad Appleton – Streamed Lines: Branching Patterns for Parallel Software Development:

<http://www.cmcrossroads.com/bradapp/acme/branching/>

This generic overview sums up a lot of patterns for branching that can be used. The process described in this process pattern is used in real life and focuses on the context as described in the section “Intent”.

8.8.4 Applicability

This process can be applied with traditional processes, but with more agile approaches as well.

8.8.5 Structure

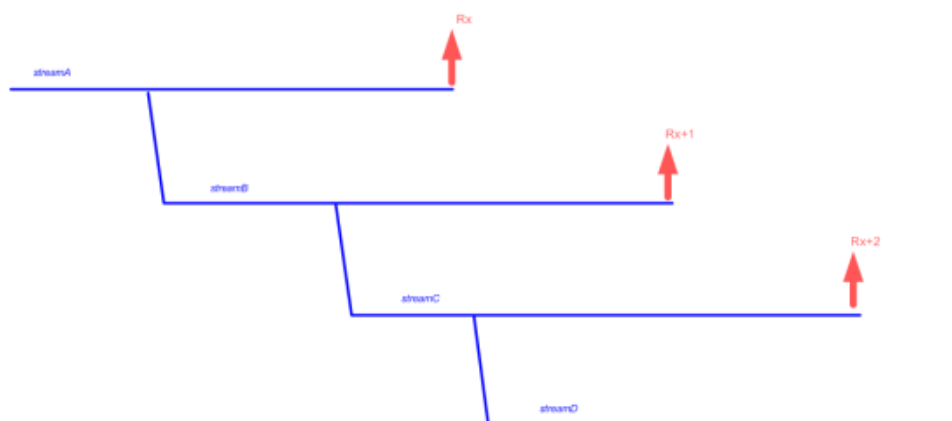
This process pattern is applied for software development of the software embedded in a medical device. Development activities in general involve hardware and software development. The software is backwards compatible to be able to supply software upgrades and repairs to existing customers.

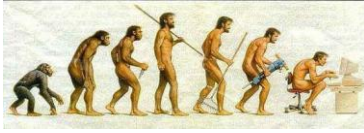
A number of basic starting points of evolutionary development:

- Evolutionary development approach: $R_{x+1} = R_x + \Delta$
 - Typically: $\Delta > 0$ – new features, additional configurations
 - But also: $\Delta < 0$ – removed ‘legacy’ functionality, removed configurations
- SW upgrade strategy: backwards compatibility
 - $R_x \xrightarrow{\text{upgrade}} R_{x+1}$
 - But not: $R_{x+1} \xrightarrow{\text{upgrade}} R_x$
- Backwards compatibility w.r.t:
 - Database datamodel
 - Protocols
 - Supported (hw) configurations
 - But also: functionality and bugfixes

The basic concept is a Cascade model. Each next software stream branches from the previous stream.

Some reasons to branch are:

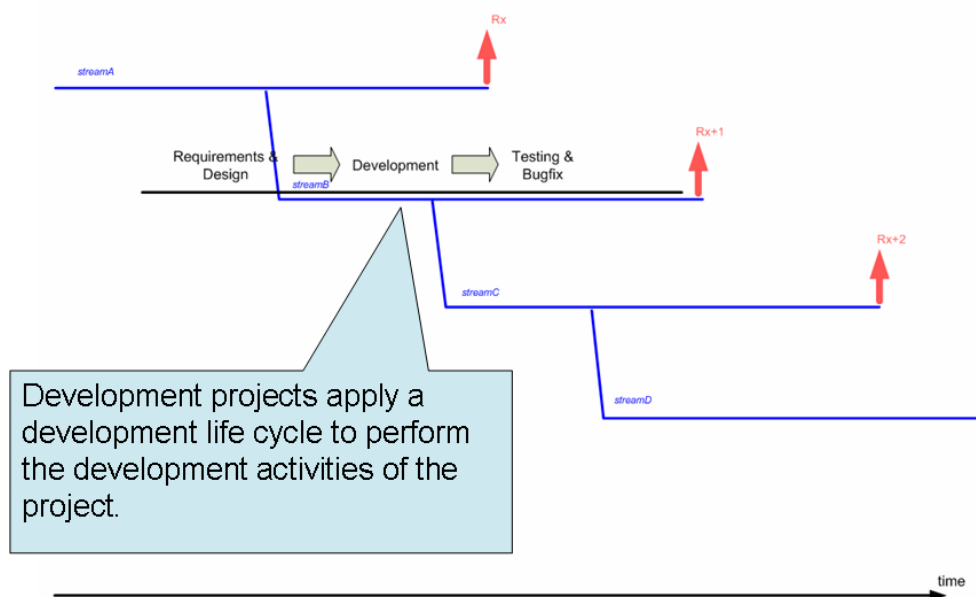




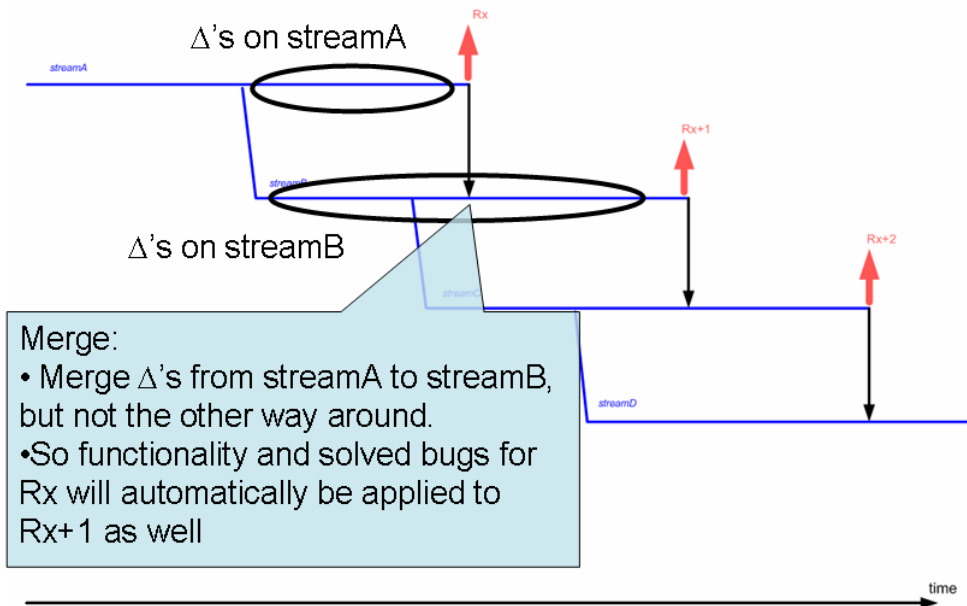
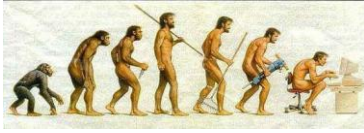
- To prevent any risks resulting from the software development on stream B to influence stream A.
- Inconsistency between the (lead time of the) development lifecycle of hardware and software.
- Business reasons imposed by the costs and availability of hardware under development.

The moment to branch is a trade-off: branching typically results in overhead (coordination, merging); for this reason the moment to branch is delayed as long as possible.

The project's development lifecycle is mapped to the software stream. To prevent unnecessary branching, more than one project can work on the same software stream. Coordination over development projects is required then; see the section on participants and collaboration.

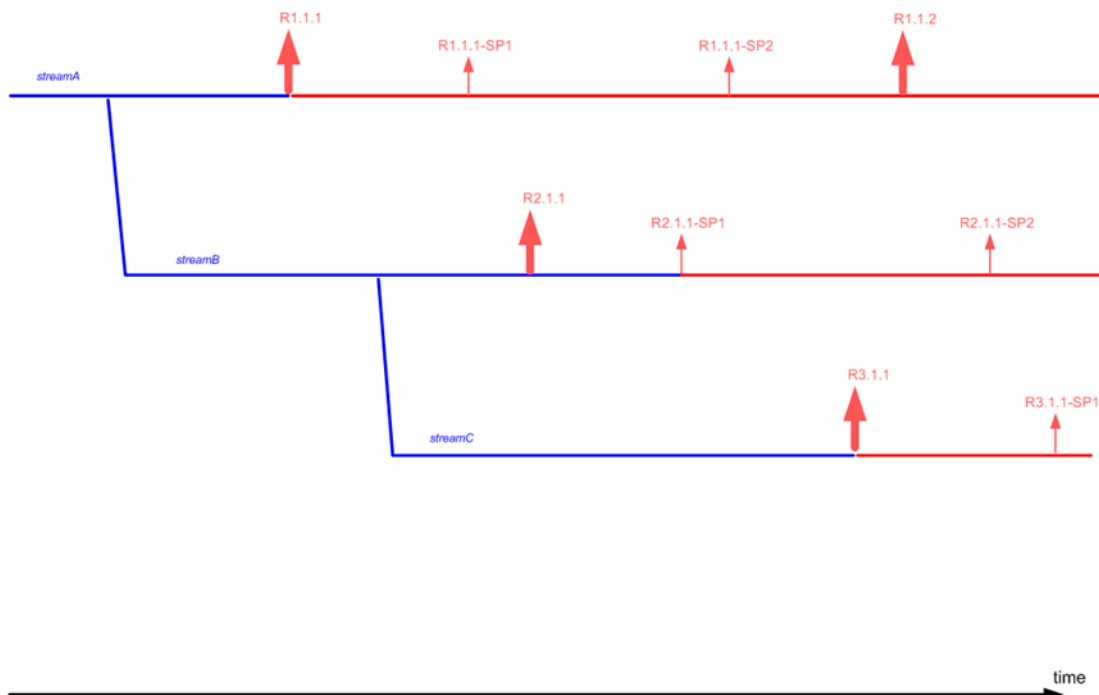


Merging typically takes place from top to bottom in this graphical representation:

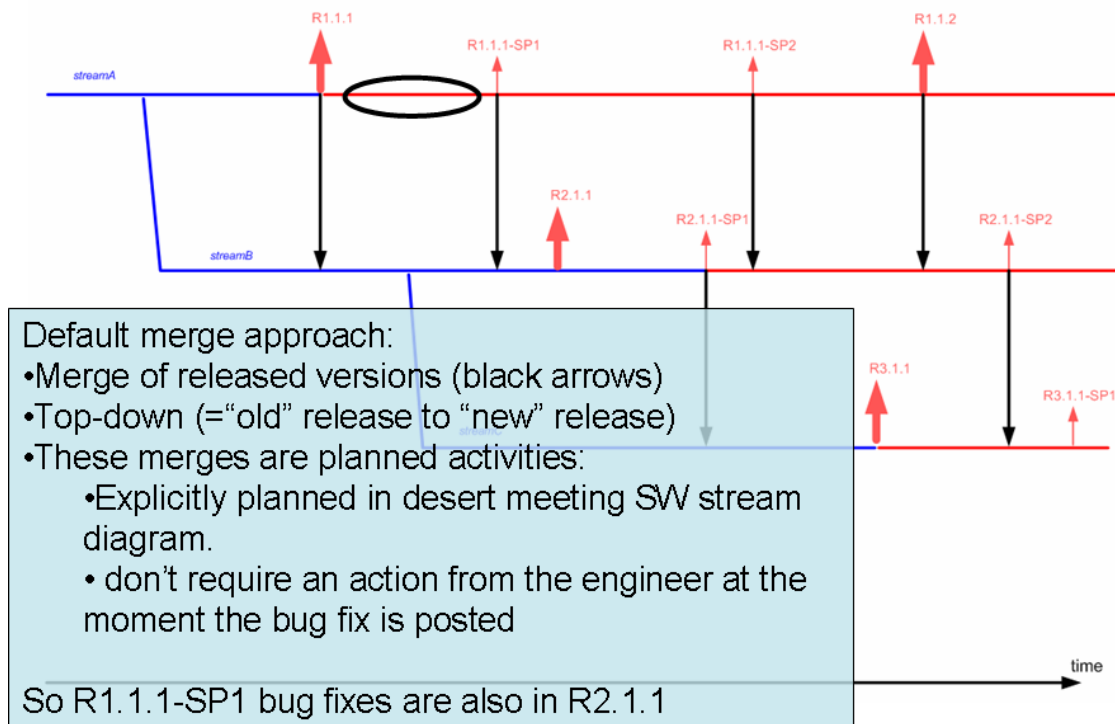
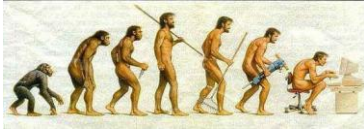


After finalizing the first major release from a software stream, the software stream goes into “maintenance mode”. From that moment onwards the stream is used to produce service packs (SP's) and levels. Levels can either bring repairs or functionality.

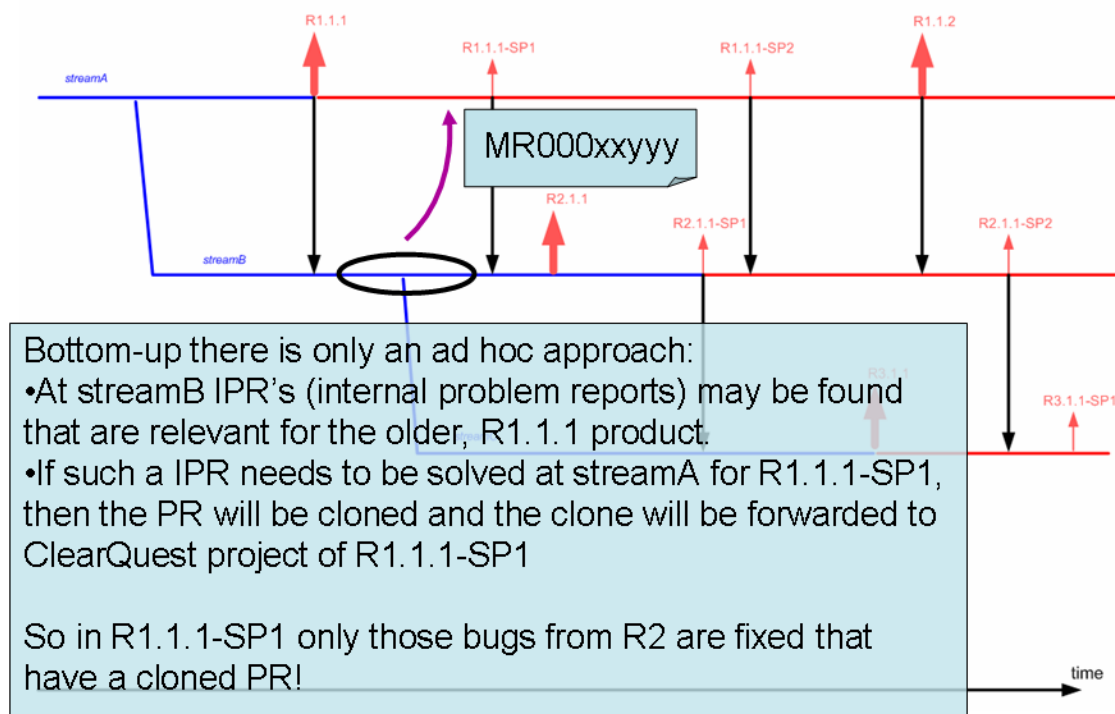
Maintenance mode is depicted in red in the next diagram:

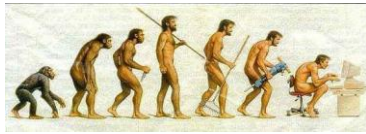


Again, default merge approach for SP's and levels is top-down:



In specific cases a “merge back” is performed to a previous release. This is always explicitly controlled via an entry in the defect database.

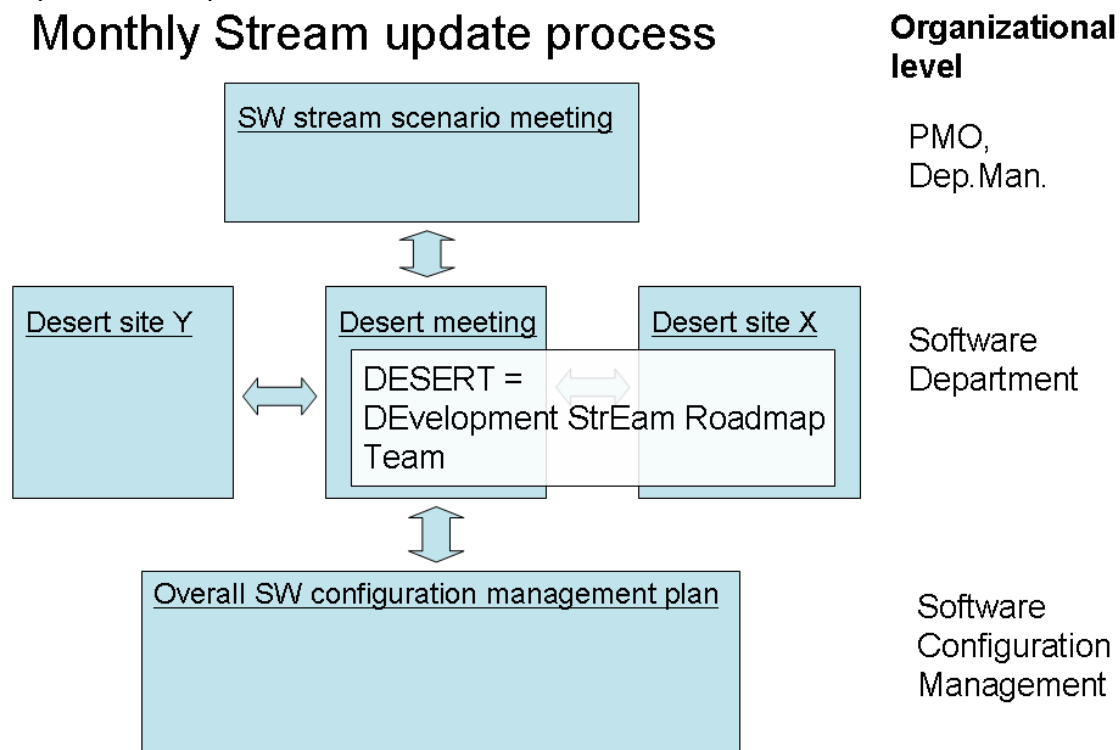




8.8.6 Participants & Collaborations

In this process there are many stakeholders: Product Program Management, Project Management, Software development at each of the development sites. Due to their nature to focus on specific project definition, projects have a tendency to sub optimize for their own project c.q. deliverables. There is a process in place to control the overall consistency over all software streams and software releases. In this process it must be possible to escalate above the interest of individual projects. The following diagram represents the process:

Monthly Stream update process



This process is executed in a monthly cycle. The SW stream scenario meeting is a mechanism to escalate over the scope over the multiple parallel projects.

The deliverable of the desert meeting is an updated diagram showing:

- All relevant software streams
- All software release level and service packs
- All merges.

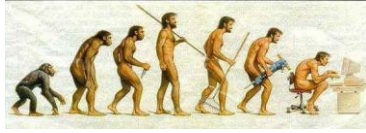
This overall picture is the basis for and overall software configuration management plan, which fills in the actual details (exact baselines, labels, stream names, release names, dates, etc).

8.8.7 Related Patterns

The Incremental Configuration Management Process Pattern is related. That process pattern focuses on components and release management. It can be considered as complementary to this process.

8.8.8 Known uses

- Philips Medical Systems



8.9 Product baseline overview pattern

8.9.1 *Process pattern name:*

Product Baselines Overview

Product baselines describe a product under development in terms of its underlying module hierarchy. The lowest module level consists of work products created by a single development discipline (such as software or electronics). Product baselines may refer to the discipline modules directly or to a collection of such modules aggregated in an intermediate baseline. To keep the discussion below simple we will consider here product baselines that consist of a single hierarchical layer of discipline modules.

The Product Baseline Overview is a history over time of the created product baselines during the development of the product.

8.9.2 *Intent*

The pattern describes practical ways of working for managing product baselines.

The outcome of this process pattern is:

- Well defined and recorded product baselines.
- It provides traceability of product features over the development history.
- It supports the change control process for the product.
- It helps manage the quality level of product baselines during the project for the different stakeholders (e.g. developers, testers, external parties).

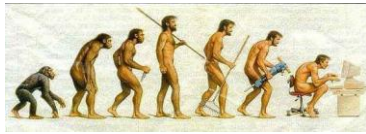
8.9.3 *Also known as*

No other name available yet.

8.9.4 *Motivation*

Products in general consist of a number of modules created by different disciplines: PCBs, mechanics, programmable logic, software. Each of the disciplines has its own development cycle with its specific tools. Often these tools support version management and poses the ability to create baselines for configuration management of the discipline work products. Different tooling is employed by different disciplines due to the specific demands of the discipline.

Configuration management at the product level, as defined by CMM-i, requires the creation and administration of product baselines. These product baselines are defined normally in terms of the discipline baselines, in other words: a product baseline is a fixed collection of specific discipline baselines. For administrating these product baselines there is a need for a simple and easy to maintain way of recording the baselines. This tool provides a solution to the following difficulties in the existing development flows:



- Product baselines are not supported by existing discipline configuration or version management tools: these tools generally do not support baseline hierarchies, and in most cases don't record baseline quality levels (with associated evidence).
- Some discipline development flows lack (full) configuration management.

Per product baseline the following items need to be recorded in order to perform useful and consistent configuration management:

- The product baseline identification.
- The product baseline contents, consisting of the baseline identifications of the composing discipline modules and the product defining documentation.
- The quality level of the baseline, including (a pointer to) the evidence on which the level is based.
- If desired an overview of the most important changes with respect to a previous baseline, including solved (major) defects.

Additionally to the product baselines the same format can be used to perform discipline baseline administration for those disciplines that lack proper version control tooling in their development flow.

8.9.5 Applicability

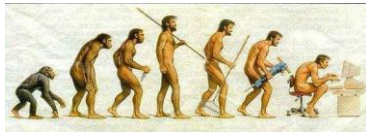
The pattern is applicable throughout the complete development lifecycle of a product.

8.9.6 Structure

The process steps of this pattern:

- The Configuration Items (CI's) that the project will deliver are clearly identified.
Properties for the definition of a CI:
 - Controlled interfaces with respect to other configuration items.
 - Own documentation structure.
 - Independent development, verification & validation.
 - Independent life cycle.
 - Form, Fit & Function replacement possible.
 - May be used by more than one users/clients.
- The content of each Configuration Item is defined.
- The way of identifying product baselines and discipline baselines is established, e.g. use of a company number system or specially constructed labels.
- A form is set up that contains fields for entering the administration items mentioned under section 4.
- Each product baseline is described by a filled instantiation of the form. The collection of filled forms constitutes the Product Baseline Overview.
- Similar forms are or may be maintained for disciplines lacking proper version control and baseline labeling.

The following figure shows a template that can be used to set up the form:



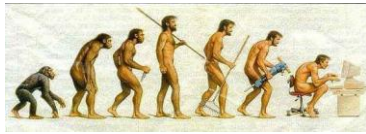
Product Baseline ID		<this column contains the field descriptions, make a column per baseline> <product baseline identification>	
Discipline baseline content:	SW CI name	identification 12 NC/<baseline id>	Content/Evidence <EC request>/<Document reference>
	HW CI name	12 NC/<baseline id>	<EC request>/<Document reference>
	HW CI name	12 NC/<baseline id>	<EC request>/<Document reference>
	Mech CI name	12 NC/<baseline id>	<EC request>/<Document reference>
Documentation baseline DHF Exceptions		<DHF form identification>	
Test and Verification <tool, script name>		<Baseline id>	<document reference>
Product Promotion			
INITIAL			
BUILT			
TESTED	date	<dd-mm-yy>	
	evidence	<reference document or tab of this worksheet>	
PUBLISHED	date	<dd-mm-yy>	
	evidence	<reference document or tab of this worksheet>	
RELEASED	date	<dd-mm-yy>	
	evidence	<reference document or tab of this worksheet>	
REJECTED			
Related Information		<optionally related information can be added in the rows below, e.g. the most important changes applied in the current baseline>	
Changes	nr / date	<dd-mm-yyyy>	
	description	<e.g. added Increment X functions>	
	nr / date	<Change Request number>	
Defects	Solved	<Problem Report number>	
	Found	<submitted PR number>	

8.9.7 Participants & collaborations

The product integration manager is responsible for the content of the product baseline overview. The actual administrative work can be performed by the project configuration manager. Discipline integrators are responsible for the delivery of discipline baseline information.

8.9.8 Consequences

Performing product baseline administration following the presented process pattern enables a project to create an overview of product evolution, changes and defects. It serves as the basis for product release bulletins and changes notes. It enables a project to step back to earlier product versions in case of severe mishap in any development phase or increment. It provides a precise content definition of delivered product versions, needed in case one or more customers experience problems with



(intermediate versions of) the product. These benefits are achieved by a consistent, sufficiently complete and timely administration of the product baselines with their properties as described by this process pattern.

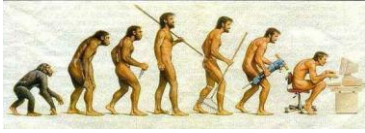
8.9.9 Implementation

A simple implementation example is based on a spreadsheet program:

- A sheet is created for each product/discipline baseline overview to be administrated.
- Each sheet has in its first column(s) a description and labeling of the applicable entry fields.
- The following columns each define a product baseline by means of the filled in data.
- Restrict any change information (if present) to the most important changes/problems to keep the overview compact and efficient.
- See the sample below.

8.9.10 Sample

Product Baseline		ID		INC2	
Discipline baseline content:		Identification		Content/Evidence	
IPB-FOX		4522 167 03904 rev. 0		see tab: IPB-FOX	
FPGA 1		4522 166 15322 / rev 0		see tab: FPGA 1	
FPGA 2		4522 166 15312 / rev 0		see tab: FPGA 2	
Documentation baseline		DHF			
Exceptions		PL1-066-07-0024 v1.0			
		none			
Test and Verification		Identification		Content/Evidence	
test scripts		INC2_23.15.23.0_070801_152113		see ClearCase	
Product Promotion					
INITIAL					
BUILT					
TESTED		date 2008-02-05			
		evidence regression tests passed, see PL1-066-07-0155, v1.1			
PUBLISHED		date 2008-02-25			
		evidence product test passed, see PL1-066-07-0155, v1.1			
RELEASED		date			
		evidence			
REJECTED					
Related Information					
Changes		nr / date		II07xxxxxx / 17-12-2007	
		description		updated SW release	
		nr / date			
		description			
Defects		Solved		CIS000xxxxx	
		Found			

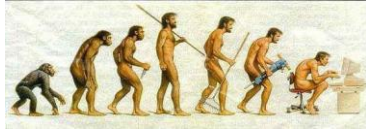


8.9.11 Known uses

- Philips Medical Systems

8.9.12 Related Patterns

- Incremental configuration management.



9 References

- [APACHE] About the Apache incubator, <http://incubator.apache.org/> , Julio 2006.
- [ECLIPSE] Validation phase, http://www.eclipse.org/projects/dev_process/validation-phase.php , Julio 2006.
- [BOEHM] ISBN 0-13-822122-7 Software Engineering Economics, B.W. Boehm, 1981
- [CAPERS] ISBN 0-07-913094-1 Estimating Software Costs, T. Capers Jones, 1998
- [HUMPHREY] ISBN 0-201-54610-8 A discipline for Software Engineering, Watts S. Humphrey, 1995
- [LEDERER] Communications of the ACM, Vol. 35, Nr. 2, February 1992, pp. 51-59
Nine Management Guidelines for Better Cost Estimating, Albert L. Lederer and Jayesh Prasad,
- [RAD] ISBN 1-55615-900-5 Rapid Development, Taming Wild Software Schedules, Steve McConnell, Microsoft Press, 1996
- [SWEST] Software Estimation Course, Centre for Technical