Module Supporting Processes

Gerrit Muller
Embedded Systems Institute
Den Dolech 2 (Laplace Building 0.10) P.O. Box 513, 5600 MB Eindhoven The Netherlands
gerrit.muller@embeddedsystems.nl

Abstract

This module addresses supporting processes, for instance documentation, templates, and reviewing.

Distribution
This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

All Gaudí documents are available at:
http://www.gaudisite.nl/

version: 1.2 status: draft July 20, 2011

Contents

1 Granularity of Documentation **1**
1.1 Introduction 1
1.2 Stakeholders 1
1.2.1 Example digital flat screen TV 2
1.3 Requirements 2
1.4 Documentation Structure 5
1.5 Payload, the ratio between overhead and content 7
1.6 Acknowledgements 8

2 LEAN and A3 Approach to Supporting Processes **9**
2.1 Introduction 9
2.2 LEAN and Supportive Processes in General 10
2.3 A3 Essentials 10
2.4 Example of an A3 11

3 Light Weight Review Process **13**
3.1 Introduction 13
3.2 The Review Process 14
3.2.1 Consultation & Review 15
3.2.2 Final Review 16
3.2.3 Authorization 17
3.2.4 Change Request handling 18
3.3 Complementary Processes 18

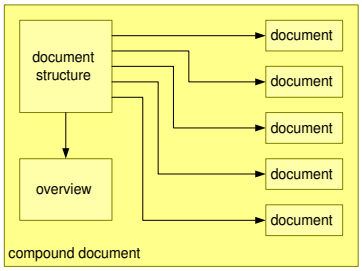
4 Template How To **19**
4.1 Introduction 19
4.2 Why Templates? 20
4.3 New Process Introduction 20
4.4 What does a Template contain 21
4.5 Copy Paste Modify Pattern 22
4.6 Template Development 23
4.7 Guidelines 23

4.8 Pitfalls 25
4.9 Why I hate templates 25
4.10 Summary 26
4.11 Acknowledgements 26

5 System Integration How-To 27
5.1 Introduction 27
5.1.1 Goal of Integration 28
5.1.2 Product Integration as part of Product Creation Process . . 28
5.1.3 Integration in Relation to Testing 29
5.2 What, How, When and Who of Integration 30
5.3 Configuration Management 36
5.4 Typical Order of Integration Problems Occurring in Real Life . . . 38
5.5 Acknowledgements 40

Chapter 1

Granularity of Documentation



1.1 Introduction

Documentation is an important communication means in the Product Creation Process. The whole documentation set is written by multiple authors with different competencies. System architects contribute to the structure of the documentation, and write a small subset of the documentation themselves. The size of the units within the documentation structure is called the granularity of the documentation.

The right level of granularity improves the effectiveness of the documentation. We discuss criteria to design the documentation structure, the documentation granularity, and the documentation processes.

1.2 Stakeholders

Figure 1.1 shows the stakeholders of a document. The document is a description of some function or component that has to be realized by means of an implementation. The producers and the consumers of the function or component are the main stakeholders of the document. The author is also an important stakeholder. The function or component is always realized and used within a broader context. This context interacts with the function or component, so the persons responsible

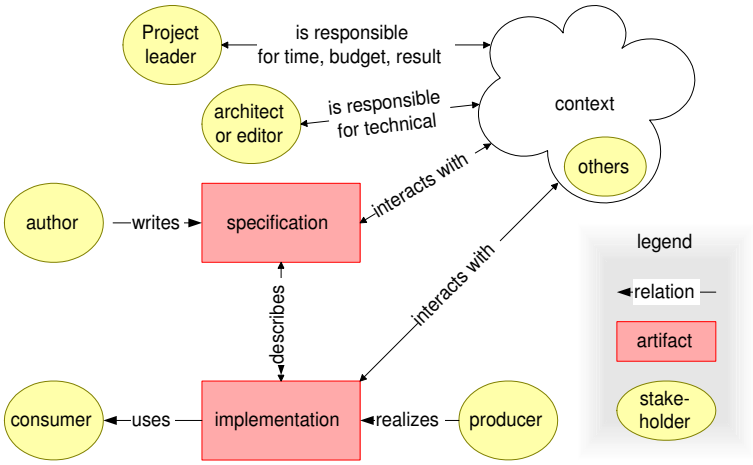


Figure 1.1: The stakeholders of a single document

for the context are also stakeholder of the document. In the context there will be other stakeholders as well; people who do have some involvement with the function or component.

1.2.1 Example digital flat screen TV

An electronics designer writes a specification for a Printed Circuit Board (PCB) to be used in a digital flat screen TV. A digital designer and a layout engineer realize the design, hence they are the producers. A software engineer will write the software making use of the functionality of the board, he is one of the consumers. The product (the digital flat screen TV) is the context for this PCB. The designer of the power supply might be a stakeholder, especially if the PCB has specific power requirements. The industrial designer responsible for the packaging is another stakeholder. The final product will have a project leader, responsible for the schedules, costs et cetera and is stakeholder with respect to these issues. The architect at last is responsible for a balanced and consistent product design, where the PCB should fit in.

1.3 Requirements

The documentation of a product need to be decomposed in smaller units, with the smallest units being atomic documents. We will discuss the requirements for the entire documentation structure, the documents itself, and the underlying process.

The criteria for the entire documentation structure and process are:

Accessibility for the readers ; the information should be understandable and readable

for the intended audience. The signal-to-noise ratio in the document must be high; information should not be hidden in a sea of words.

Low threshold for the readers ; No hurdles such as many pages of meta information, cumbersome security provisions, or complicated tools should dissuade readers from actually reading the document

Low threshold for the authors ; authors have to be encouraged to write. Hurdles, such as poor tools or cumbersome procedures, provide an excuse to delay writing.

Completeness of important information. Note that real completeness is an illusion, there are always more details that can be documented. All crucial aspects have to be covered by the entire documentation set.

Consistency of the information throughout the documentation. The writers strive for consistency, but we have to realize that in the complex world with many stakeholders some inconsistencies can be present. Inconsistencies that have significant impact on the result have to be removed.

Maintainability of the entire documentation, both during product creation as well as during the rest of the product life cycle.

Scalability of the documentation structure to later project phases, where many more engineers can be involved. The following measures help for scalability:

- well defined documentation structure
- explicit overview specifications at higher aggregation levels
- recursive application of structure and overview documents
- distribution of the review process

Evolvability of the documentation over time. Most documentation is re-used in successive projects.

Process to ensure the quality of the information . The quality of the content of the information is core to good results. Documentation that has been made only to satisfy the procedure is a waste of effort and time.

From reader point of view this translates in the requirements for the document infrastructure: it must be fast and easy to *view* and to *print* documents, and *searching* in the documentation also has to be fast and easy. Searching must be possible in a structured, e.g. hierarchical, way, and also via free text “a la Google”. Any part of the documentation must be reachable within a limited number of steps, so no excessively deep document hierarchies.

The criteria for the documents within the documentation structure are:

High cohesion within the document. The information in a document has to "belong" together. If information is not connected to the rest of the document, then this information might belong in another document.

Low coupling with other documents. Some coupling will be present, since the parts together will form the system. If the coupling is high, then the document decomposition is suspect and might need improvement.

Accessibility for the readers, as for the entire documentation.

Low threshold for the reader, as for the entire documentation.

Low threshold for the author, as for the entire documentation.

Manageable steps to create, review, and change the document. Documents in product creation are reviewed and updated frequently. Hence these operations should take limited effort and time. The consequence is that single documents should not be large.

Clear responsibilities, especially for the content of the document. Documents with multiple authors are suspect, responsibility for the content can be diffuse. Worse are documents where an anonymous team or committee is "the author". If a document needs multiple authors, then it is often a symptom of bad decomposition. Also the reviewers responsibility must be clear, hence we recommend to limit the number of reviewers. When many reviewers are needed, then the decomposition is again suspect.

Clear position and relation with the context documents only make sense in the intended context. On purpose the information is captured in multiple documents. Therefor for every individual document it should be clear in what context it belongs and how it relates to other documents.

Well-defined status of the information. Documents are used and most valuable in the period when they are created. The content can be quite preliminary or draft. The document must clearly indicate what the status is of its content, so that readers can use it with proper precautions.

Timely availability of the document. When documents are too late available we do not harvest the value. Authors have to balance quality, completeness,and consistency against the required effort and time.

A very important function of documentation is communication. Communication requires that the information is accessible for all stakeholders, and that the threshold to produce documentation or to use documentation should be low[1].

[1] Quite often organizations focus on the documentation procedures, and documentation

1.4 Documentation Structure

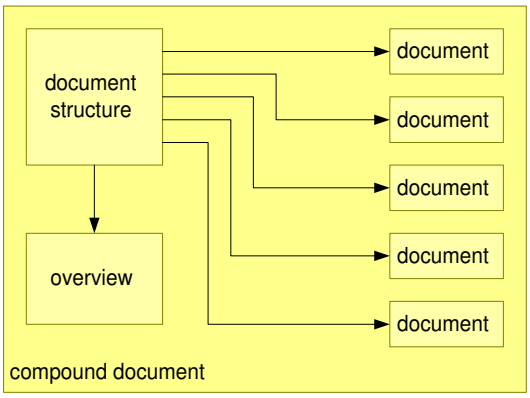


Figure 1.2: Large documents are decomposed in smaller documents, supported by a document structure and overview

The standard way to cope with large amounts of information is to decompose the information in smaller parts. The decomposition of the large amount of information results in a set of smaller documents. The structure of such a decomposition is made explicit in the "documentation structure", fulfilling the requirement to have a *well defined documentation structure*. The documentation structure is managed as a normal document. An overview document is required to keep the overview accessible, addressing the requirement to have *overview specifications at higher aggregation levels*. Overviews help the readers, especially when the more detailed information gets scattered in smaller documents.

This decomposition is applied recursively, see Figure 1.3. In this way the granularity supports the realization of the requirements as described in the 1.3. For instance, the principle of *recursion* is a good answer to the requirements related to *scalability* of the entire documentation. Creating explicit structure and overview documents and allocating creation and maintenance to authors supports *maintainability*.

A fine grain structure, e.g. small documents, lower the threshold to make documents and to read the contents, in this way answering document requirements *accessibility for the reader*, *low threshold for the reader* and *low threshold for the author*.

The clarity and the value of the content is the foremost requirement for documentation. Decomposing the documentation is a balancing act in many dimensions, similar to the decomposition of systems. Clarity and value of the content may not

management, forgetting the main drivers mentioned in this subsection. The result can be tremendous thresholds, causing either apathy or bypasses. It cannot be stressed enough that procedures and tools are the **means** to solve a problem and not a goal in itself

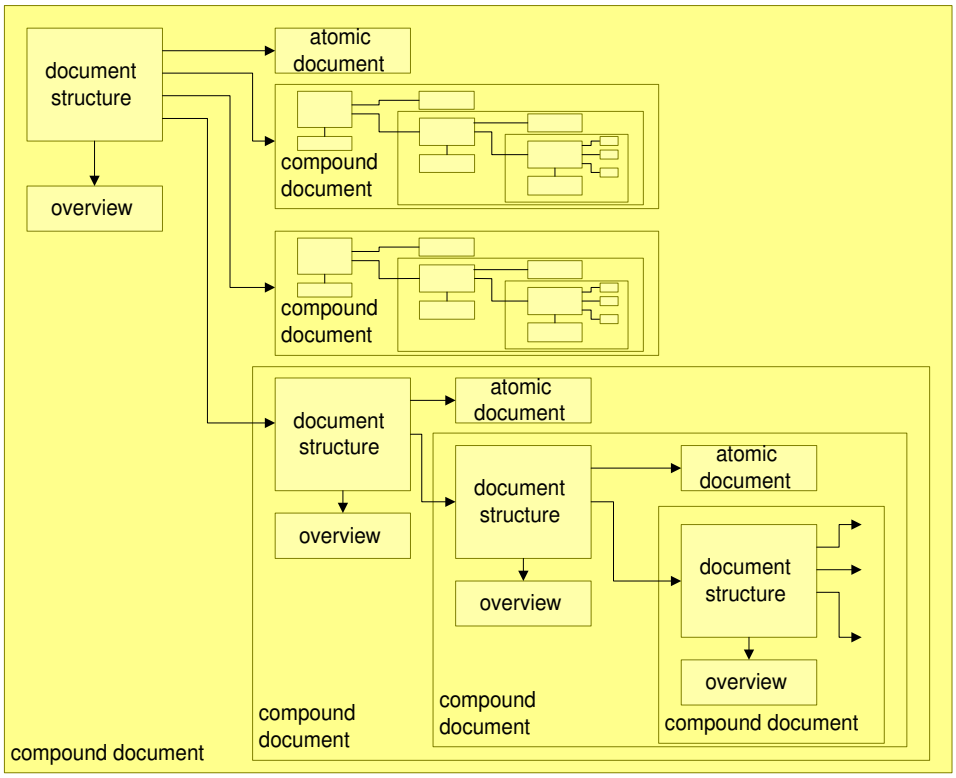


Figure 1.3: Decomposition is applied recursively until the atomic documents fulfill the requirements in section 1.3

suffer from the structure. Dogmatic structuring rules might be conflicting with clear responsibilities (*single author*). When authors write outside their expertise area, then there is a severe quality risk. The decomposition has to result in sufficiently small documents to support the requirement *Manageable steps to create, review, and change*, Large, monolithic documents violate this requirement.

The document granularity is an important design criterion for the documentation structure. The extreme that every *single value* is an entity[2] is not optimal, because the relations between values are even more important than the value itself. In case of *single value* documentation, relations are lost. The other extreme, to put everything in a single document, is conflicting with many of the requirements, such as *manageability*, *clear responsibilities*, *well-defined status* and *timely availability*. The granularity aspect, with the many psychological factors involved, is further

[2] A common pitfall is to store all values in a database. In this way every value is an entity in itself. Such a database creates the suggestion of completeness and flexibility, but in reality it becomes a big heap, where the designers lose the overview. These databases may help the verification process, but do not fulfill the documentation needs.

discussed in 1.5.

1.5 Payload, the ratio between overhead and content

An atomic document must be small enough to be accessible to readers. Thick documents are put on top of the stack of "interesting papers to be read", to be removed when this stack overflows. For most people time is the most scarce resource. Struggling through all kinds of overhead is a waste of their scarce and valuable time. Documentation effectively supports communication if the reader can start directly with reading the relevant information. Figure 1.4 shows the layout of a good document.

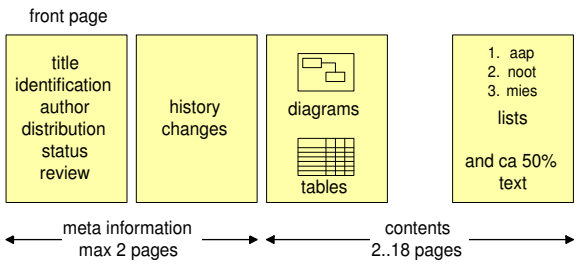


Figure 1.4: Layout of a good document, heuristic for the number of pages of a good document is $4 \leq nrofpages \leq 20$

The front page is used for all relevant meta-information. Meta-information is the information required for the document management, defining the status, responsibilities, context etc. The history and change information on the second page should be a service to the readers, to enable them to quickly see the relevant changes relative to earlier versions they might have read. More extensive change information, required for quality assurance purposes can be present in the document management system, it should not distract the reader from the information itself.

Such a document needs only to be opened to access the contents. Many older organizations tend to make documents with up to 10 pages of overhead information. Many people are interrupted by phone, calendar, e-mail, or person before reaching page three. The overhead de facto inhibits people to read the contents of badly written documents[3].

The contents of a well written document ought to be optimized to get the essential information transferred. The reader community exists of different people, with differing reading and learning styles. To get information across the information must be visualized (diagrams), structured and summarized (tables and lists)

[3]Often the situation is much worse than described here. In name of "standardization" these counterproductive layouts are made mandatory, forcing everyone to create thresholds for readers!

and, to a limited extend, explained in text.

Once a document start its life cycle, the next risk is that the document keeps growing Authors have the tendency to transform comments and critiques of readers in explaining text. Unfortunately, large sections of text hide the key information, and violation of the maximum of 20 pages gets probable. It is better to translate the comments and critiques back into an improved diagram, table or list. Authors have to find the root cause of reader comments. For example an unclear diagram gives rise to misunderstanding.

Another frequent occurring trap is the extension of a document with missing context information. For instance, if the higher level specification is missing, parts of that specification are included in the lower level specification. An effective counter measure for this trap is to write the specification structure, showing the context and enabling to write the context later step by step. This strategy results in documents that are more focused, have a better cohesion internally, and have less coupling with other documents.

The heuristic mentioned in Figure 1.4 is that a good document should have 4 or more pages. This minimum should trigger people with the question if the information in a very small document has a right of existence on its own. The ratio overhead versus payload for very small documents is unbalanced. There are a small documents were the small size is appropriate.

The maximum number of pages for a good document is 20. These documents don't scare people away yet. A 20 page document can be read in less than one hour, and the review can also be done in less than one hour. For many purposes 10 to 15 page documents are optimal. If documents require more than 20 pages the recipe is simple: make it a compound document, so split the content in multiple smaller documents.

In large documents a natural split up is often directly visible.

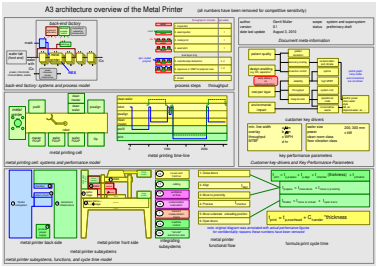
Large documents often violate a number of the requirements in 1.3. For instance, the document is edited by a single person but written by multiple authors. Another symptom of requirement violation is a document that is partly finished and partly in draft status (for instance"requirements" sections are written, while the "design" is still in full motion).

1.6 Acknowledgements

Angelo Hulshout triggered me to fill the the open ends in the requirements section.

Chapter 2

LEAN and A3 Approach to Supporting Processes



2.1 Introduction

LEAN manufacturing is a manufacturing approach based on Toyota's successes, as described by researchers who observed and analyzed Toyota. LEAN product development is building on LEAN manufacturing, where the ideas from the repetitive production environment are transformed for use in the creative product development environment. Likewise LEAN product development is based on observing Toyota product development.

LEAN at slogan level is sold as "avoid waste". For the purpose of this chapter we characterize LEAN by the following elements, loosely based on [2]:

A holistic, systems approach to product development including people, processes, and technology.

Multi-disciplinary from the early start, with a drive to be fact based.

Customer understanding as the the starting point.

Continuous improvement and learning as cultural value.

Small distance between engineers and real systems, including manufacturing, sales and service and the system of interest.

2.2 LEAN and Supportive Processes in General

LEAN product development delegates responsibilities as much as possible to the experts. The management facilitates and stimulates the experts to operate towards the goals using the LEAN principles. The way of working is highly pragmatic, where the goal dominates over the means. In many cases no complicated computer tools and repositories are used.

Co-location in a larger room is common. In this room wall space is used to visualize plans and schedules, with low-tech means such as paper, pens, or magnetic boards. The components or a prototype of the system can be present in the room (keep the distance small!).

In LEAN manufacturing and LEAN product development A3's are used to document and communicate, as discussed in the next section. A3 is an European standard paper size of $297 * 420mm$.

2.3 A3 Essentials

An A3 contains a "human friendly" amount of information. The size permits some depth and facts in the information, while at the same time the size forces the author of the A3 to select and process the information carefully.

We have the following guidelines when using A3's as the unit of documentation and communication:

Capture "hot" topics that are currently under discussion. when topics are under discussion, then explicit diagrams facilitate the discussion. The active use of the A3 will stimulate the evolution of the A3 itself.

One topic per A3 so that every A3 is homogeneous. The requirements for documents, defined in 1.3 also apply for A3 documents.

Show multiple related views. The strength of the A3 format is that several diagrams can be shown at the same time. These diagrams are different views on the same topic. These views will be related. These relations should be present in supportive, non-dominating, way, e.g. by the use of colors, shapes, lines, labels, or naming conventions.

Make the A3 digestable by limiting the amount of content. Note that the size limitation forces the authors to limit the amount of information.

Make the diagrams and information specific, for example by quantification. Note that the risk of the size limitation is that too "empty" or too glossy posters are made. Good A3's have substance; specific information helps to make the A3 substantial.

Practical visualizations close to the experience of the stakeholders. Good A3 documents engage the stakeholders helped by instant recognition of the visualizations.

Note that the granularity and structuring guidelines of 1 are applicable on A3 based documentation as well, where the pay load size is limited by the A3 dimensions.

2.4 Example of an A3

Figure 2.1 shows an example of an architecture overview shown on a single A3. This A3 shows the "super-super" system: the wafer back-end factory, where nearly finished wafers are processed and where Integrated Circuits (ICs) are produced. Part of the process at factory level is the metal printing. The metal printing related process steps are shown at factory level, both visually, as work flow steps, as well as quantifying the throughput in minutes per wafer.

The next layer in Figure 2.1 shows the "super" system: the cell. In factories all equipment related to a process step is organized in cells. A cell is a self sustained unit in the factory that can perform all operations required for this specific process step. The core entity of the cell is the wafer handling robot. This robot transports wafers from the containers with wafers (so-called FOUPs) to the functions in the cell, such as pre-fill, clean and print. The flow of the wafers through the cell is visualized at the right hand side for one master and one wafer. The previous and next wafers are simultaneously in the cell; the cell is processing wafers in pipelined mode.

The third layer shows the decomposition of the metal printer, the system of interest. These subsystems are shown as back side and front side views plus 7 integrating subsystems. Next to the subsystem decomposition the work flow of the metal printer is shown. This work flow is used to create a simple cycle time model as formula. Note that in the original A3 the formula was annotated with actual performance numbers to provide numerical insight in the cycle time.

At the top right hand side of the A3 a customer key driver graph is shown and below the graph the key performance parameters are summarized.

This single A3 shows the system, the system context and the first level of decomposition. Physical views and functional views are shown. Quantifications are given at all three levels as time-line, as table or as formula.

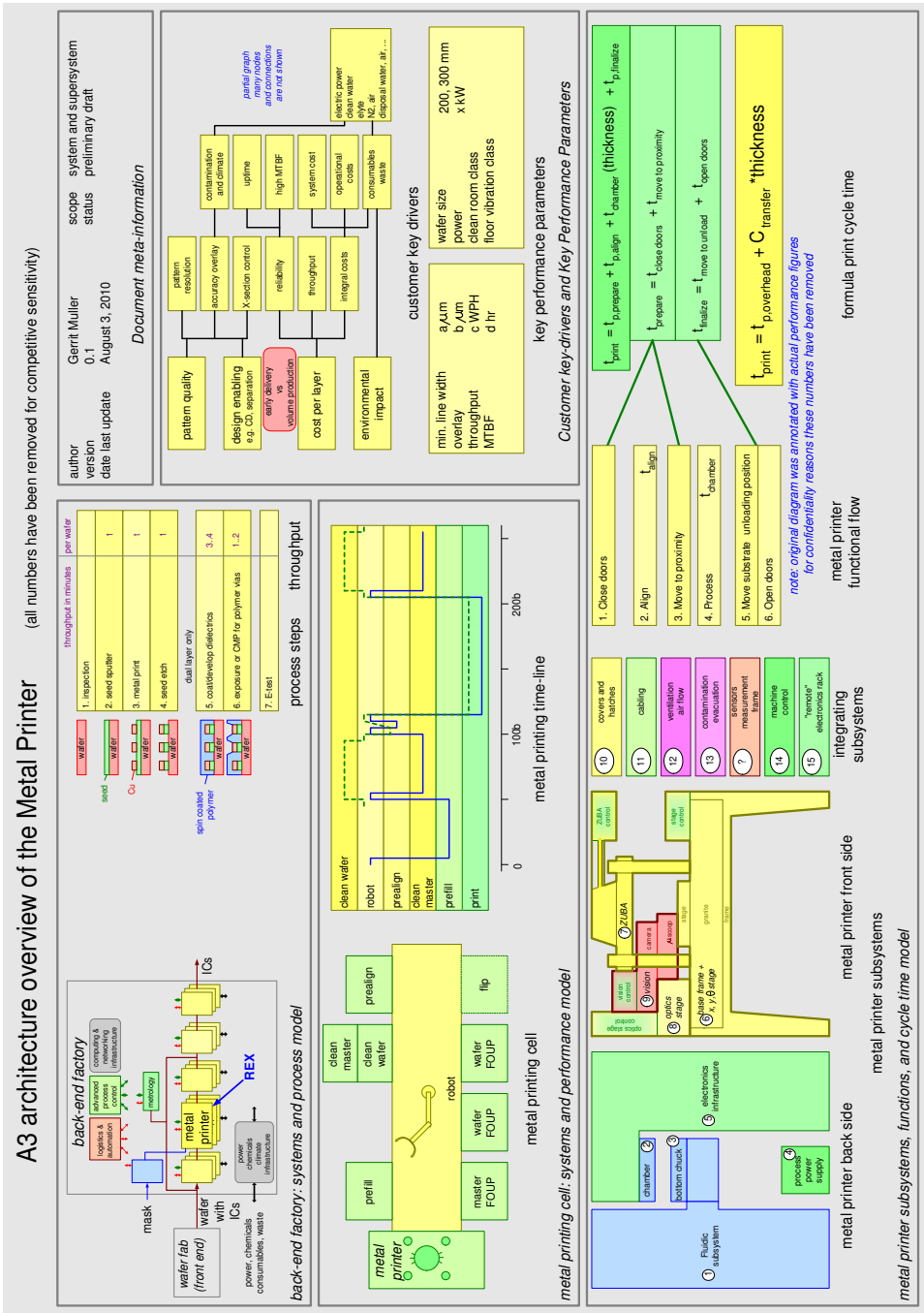
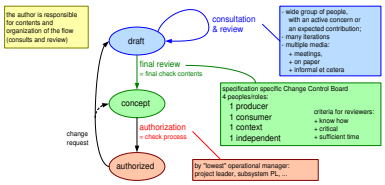


Figure 2.1: Example of an Architecture Overview on one A3

Chapter 3

Light Weight Review Process



3.1 Introduction

The creation of a product is a rather dynamic happening, full of uncertainties and with a lot of time and cost pressure. During the creation many documents are created and updated describing the product and it's design, ranging from product specifications to detailed design and test specifications. Later in the life cycle also a lot of documents are used and maintained, the so-called *Technical Product Documentation* (TPD). The TPD is the final delivery of the product creation process. The TPD is much more stable than the documents used during the creation. The TPD is only changed if there are manufacturing or logistics problems, such as components that are end-of-life, or for safety, security or reliability problems in the field. This document describes the review process for documents in the product creation process.

Figure 3.1 shows the product life cycle and the related change management processes. During the product creation phase a project team is active to create the product. This project team will discuss and implement the required changes. In fact product creation is a continuous flow of many changes. The management of these changes is kept as local as possible, by means of *micro* Specification Control Boards (SCB). In the later phases of the life cycle, during production and even after the production has stopped, a maintenance control board (MCB) handles the changes. At this time the project team has disappeared, it's members are active in new product creations.

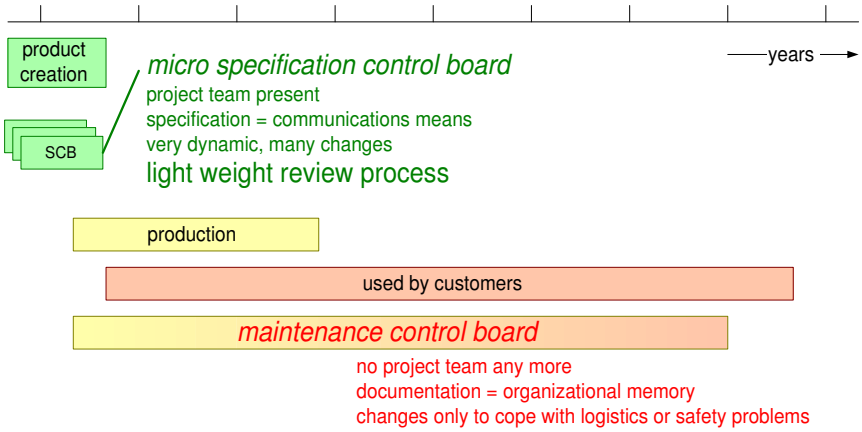


Figure 3.1: Product Life Cycle and Change Management

3.2 The Review Process

Figure 3.2 shows the state diagram of the proposed light weight review process. Only three states are present: *Draft*, *Concept* and *Authorized*. The main value of a document is during the *Draft* phase, when many decisions are taken. The document serves during the *Draft* phase as a means for communication. When the document gets more stable a more systematic final review is performed. During the final review the contents of the document is screened by a small group of reviewers. The purpose of the final review is to bring the document in the *Concept* phase, which means that the main stakeholders have a high confidence in the document contents. Finally the process of creating the document is signed off by the responsible operational manager. After sign off the document is *Authorized*. An authorized document can only be changed by a somewhat more heavy change request process. This change request process is a repetition of the same phases with the same players.

The author of the document is the owner of this process. It is the author's responsibility to create a document with the right content. The author must involve all concerned stakeholders and must organize the progress of this process. The operational manager is safeguarding the process: capable author, involvement of all relevant stakeholders, progress fitting in the project plan. This safeguarding is formalized by the Authorization, but the project leader will have to coach and monitor the author from the beginning.

In the Subsections 3.2.1, 3.2.2, and 3.2.3 the phase transitions are discussed in more detail.

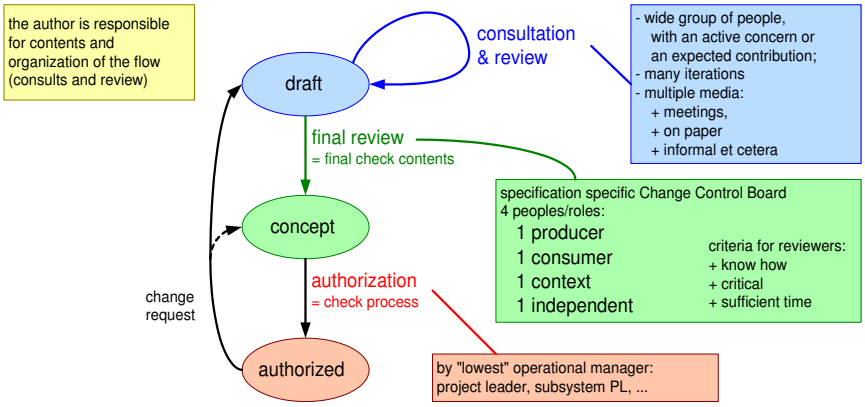


Figure 3.2: Light Weight Specification Review Process

3.2.1 Consultation & Review

Many people are involved during the consultation. Everybody with an active concern or who can make a contribution should be involved. Normally many iterations are needed for a converging specification. During the *Draft* phase decisions have to be formulated and can then be discussed by the different stakeholders. This communication can be done in many complementing ways, such as:

- design teams meetings
- specific ad hoc meetings
- bilateral discussions
- paper or electronic exchanges

Recommendations for Consultation & Review:

- Stimulate comments: approach especially those involved people, which have a deviating opinion
- The commentators (the people who are consulted) are the customers of the author; make their life easy:
  - formulate sharp questions, highlight disputes
  - communicate relevant comments to other commentators
  - maintain an accessible history, with the relevant changes
- Process comments professionally, comments can be rejected
- Communicate rejection to the relevant people, especially the originator

3.2.2 Final Review

The final review is performed by a very small, "micro" specification control board (SCB). The SCB is responsible for the right content of the specification, during the product creation process. The SCB will review the *draft* specification when the contents is sufficiently stable. However, the SCB will also review change requests once the document is authorized. The final review should not be done too early, because than a lot of overhead is created by all the change requests needed to mature the document. The status of the information in the *draft* document should be clear, because most documents are already heavily used during this phase[1].

The size of the SCB is kept small. A small team is clearly accountable. The author or project leader can look all SCB members in their eyes to see if they really did their work. In larger teams a lot of escape room is present: "I thought that my neighbor would read the specification". Input for the final review may come from all involved stakeholders, but the final review itself is performed by this small accountable SCB.

The members of the SCB must be picked with care. First of all, four different roles are recognized and should normally be present:

- producer, someone who will create what is described in the specification (not the author).
- consumer, someone who will use what is described in the specification.
- context guardian, for instance an architect, someone who is responsible for the integrity, consistency and balance of the overall design.
- independent reviewer, for instance a line manager, someone who has not been involved in the project so far and who has an independent view on the specification. This reviewer should detect blind spots.

Second, the members should have the following characteristics:

- have sufficient *know how* of the subject
- be critical
- have sufficient time available to review

These characteristics might sound trivial. However, many people prefer to avoid the load mouthed critics and invite the more silent and polite people. The danger is that real concerns are not discussed, because the too polite reviewer does not want

[1] Most parts of the system design have mutual dependencies. It is an illusion to assume that specifications can be made in a need sequential way. It is more effective to cope with the mutual dependencies by making the people aware of them and by making the dependencies explicit as much as possible

the confrontation with the author. Note that the author does not have to comply with all comments made by critical reviewers. The author may on purpose decide to keep the specification unchanged. The critical reviewer can then escalate such an issue to the project leader, or can decide to leave the issue as is. Also the time criterion sounds trivial, but in many cases project leaders want to be involved in the content discussion as well. While most project leaders are so busy that they are then the sole bottleneck in the review progress.

Some remarks and recommendations for the final review:

- The final review is a systematic check of the contents
- Preferably a meeting of the author with the change control board
- If significant changes are needed, the status stays *draft* an the review has to be repeated later.
- Distribute the document version to be reviewed to all reviewers and collect comments before the final review.
- For the consumer select the most direct or most critical user.
- The role of the Author is:
  - driver of this process
  - entry point for comments and change request
- The role of the Operational manager is:
  - entry point for escalations
  - final responsibility for the outcome (=timing and quality)

3.2.3 Authorization

The authorization is the task for the responsible operational manager, for instance the project leader. Specifications belong to the deliverables of a project, hence the project leader is responsible for the authorization

The following checklist supports the authorization:

- Has the author the right skills and know how?
- Did the author consult the right people, with the necessary information?
- Is the set of reviewers OK?:
- Is the mix of the SCB OK (producer, consumer, context, independence)?

- Do the members of the SCB have the right characteristics: know how, critical attitude, and sufficient time and opportunity?

3.2.4 Change Request handling

All documents during product creation are subject to change. Changes made during the draft phase are discussed during the ongoing consultation. However, once the document is reviewed and authorized, the expectations of the stakeholders is that they can build on the specification. If an authorized document is changed than a change request is issued. This change request temporarily causes the document to be in the *draft* phase again. After consultation, is the change accord, the SCB reviews the change also. This whole review cycle can be done with little time and effort: broadcast the change request to the involved stakeholders, and ask for approval by the SCB. The SCB approval can be issued informally, the authorization is the formalization step.

3.3 Complementary Processes

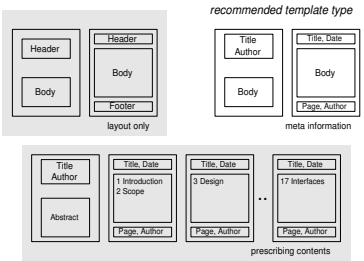
The *Light Weight Review Process* as described here is relatively informal. This review process can be complemented by more formal techniques, such as *inspections* and *audits*. A good overall balance uses light weight processes where possible, in order to focus the more costly formal approaches for the critical aspects.

Quality assurance departments or external agencies organize audits. In general audits tend to focus on the process, not on the content. Auditors look at the procedures and plans, verify the skills of the involved people, and verify dates and signatures. If the light weight process described above is part of the process description, then auditors will look at the authorization criteria, as described in Subsection 3.2.3.

Inspections can be organized by the product creation team itself. Inspections are a more systematic and formal way to go through a specification. A good inspection goes in depth and takes time. Inspections are especially suited for critical functions or requirements, such as safety and security. For more extensive reading see the book by Thomas Gilb[1]. Although the title mentions software, the same principles can be applied on systems.

Chapter 4

Template How To



4.1 Introduction

The introduction of a new process (way of working) is quite often implemented by supplying ready-to-go tools and templates. This implementation serves mainly the purpose of a smooth introduction of the new process.

Unfortunately the benefits of templates are often canceled by unforeseen side-effects, such as unintended application, inflexibility and so on. This intermezzo gives hints to avoid the **Template Trap**, so that templates can be used more effectively to support introduction of new processes.

Templates are used for all information based entities, such as documents, mechanical CAD designs and SW code. The information in this document applies to all these categories, although the text focuses on document templates.

4.2 Why Templates?

The rationale behind the use of a template is:

- Low threshold to apply a (new) process (1)
- Low effort to apply a (new) process (2)
- No need to know low level implementation details (3)
- Means to consolidate and reuse experiences (4)

Some common false arguments are:

- Obtain a uniform look (5)
- Force the application of a (new) process (6)
- Control the way a new process is applied (7)

Argument 5 is a bogus argument, uniformity[1] is not something to strive for, see section 4.9. Arguments 6 and 7 are the poor man's solution for lack of leadership and signals a dangerous disrespect for the target group.

4.3 New Process Introduction

Process Improvement drives result in enforcing existing processes or introduction of new processes. Any change introduces reactionary behavior $(action = -reaction)$, urging the process improvement people to introduce the change in such a way that this reactionary behavior gives a minimal damage, see figure 4.1.

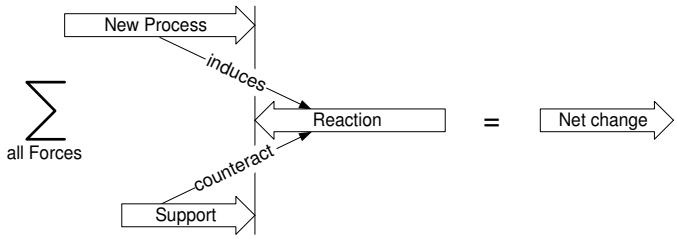


Figure 4.1: The reactionary force induced by the proposed new process is countered by giving support

The most frequent way to introduce a new process is to supply the means for the implementation of the process, in other words the emphasis is on the **how**, not

[1] This will be elaborated in a future Intermezzo, *The Uniformity Trap*.

on the **why** nor on the **what**. Figure 4.2 shows the relation between a process and a template. The process itself focuses on **why** and **what** (and who and when), while the procedure, the tools and the template are the **how**.

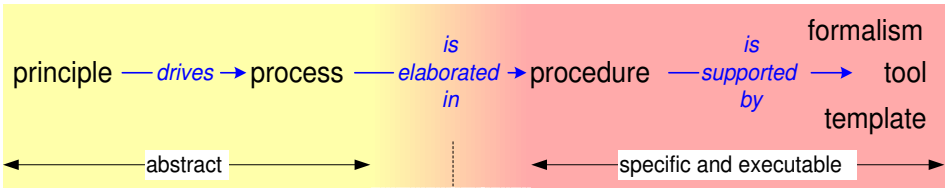


Figure 4.2: The relation between a template and a process

4.4 What does a Template contain

A template can support from *layout only* up to *complete contents standard*. Figure 4.3 shows a number of examples, Table 4.1 summarizes the characteristics. A layout only template does not have any notion of the information which will be in the document, nor does it presume anything about the process in which it is applied.

A template with meta information supports the process in which it will be applied. The template has knowledge of the meta-information of the document and supports both the layout of the document as well as the presentation of the meta-information in this layout.

template type	context knowhow	value
layout only	no	low
meta information	process	high
prescribing content	process and domain	constraining

Table 4.1: *Overview of Template characteristics*

A template which prescribes the structure of the contents of the document has knowledge of the domain as well.

Recommendation: Use a template for layout and a minimum meta information set.

Avoid using a template to structure the contents. A documentation structure needs to be **designed**, see [3]. To help people in this design process of documentation guidelines containing checklists are effective. Templates invite people to generate "noisy" chapters (which should not have been present at all), or to write monolithic documents (because the entire checklist is present in one template). Guidelines with checklists at the other hand only mention contents, without suggesting any modularity yet.

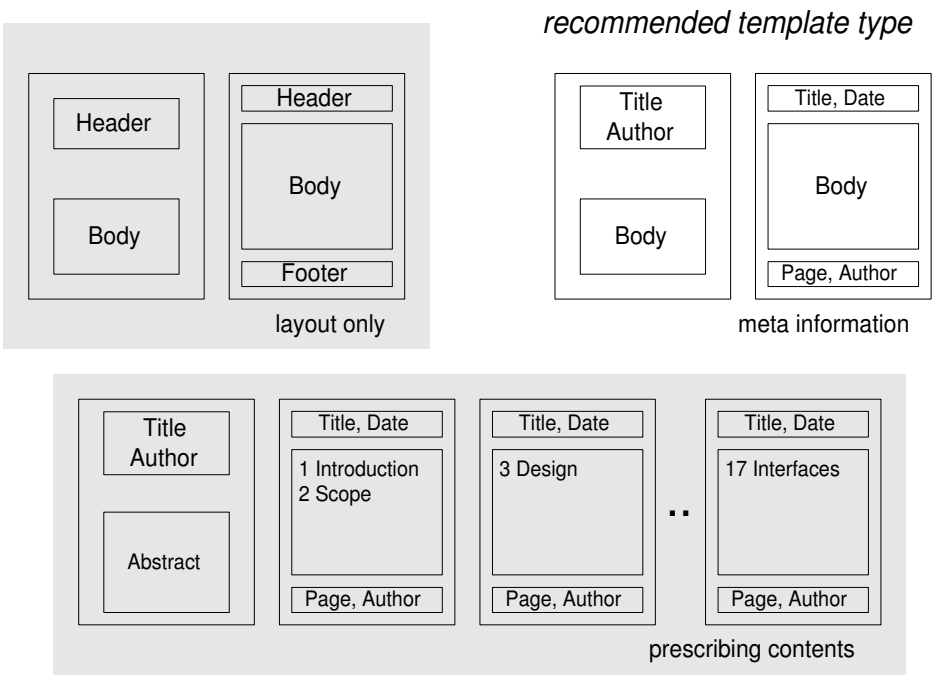


Figure 4.3: Templates from layout only, up to prescribing structure or contents, templates to support meta information are recommended.

Recommendation: Use checklists for structure and contents.

4.5 Copy Paste Modify Pattern

The understanding of the *copy paste modify pattern* will help to use templates effectively. The dominant implementation[2] strategy is the *copy paste modify pattern*:

- Look for a similar problem
- Copy its implementation
- Modify the copy to fulfil the new requirements

Majority of the work is to select the parts to be copied (or remove the unneeded parts) and to substitute the problem specific names, variables, functions et cetera.

A template is an optimization of this pattern in case of frequently reused implementations. The selection is performed once and the substitution is prepared to be easy.

[2]This holds for all information based implementations, from mechanical CAD drawings to management spreadsheets

4.6 Template Development

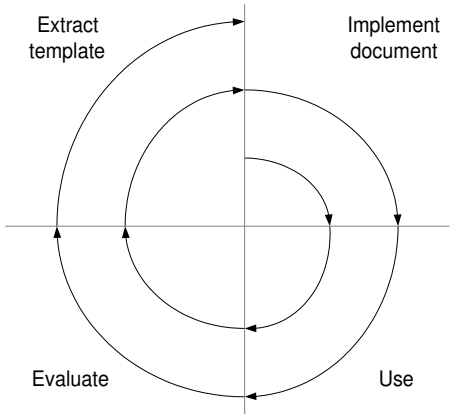


Figure 4.4: Spiral development model for Templates

The development of templates is basically the consolidation of experience. Once more a spiral development model best fits on the capturing of learning experiences, see figure 4.4. The motto is:

Use before Re-use

So implement a limited amount of documents (ca. 3), use these documents with other people, evaluate explicitly and extract then the template from these first documents. Implement the next set of documents on the basis of this template and repeat the same use, evaluation and extraction process. Keep repeating this forever!

4.7 Guidelines

A template is applied to support one or more processes. The deployment of the template is enabled by guidelines describing the way it should be used. The guidelines must be classified in mandatory rules and recommended practice.

An example of guidelines for meta information of a document is:

Mandatory on every page

- Author
- Title
- Status
- Version
- Date of last update
- Unique Identification
- Business Unit
- Page number

Mandatory on every document on top of the mandatory information per page

- Distribution (Notification) list
- Reviewers and commentators
- Document scope (Product family, Product, Subsystem, Module as far as applicable)
- Change history

Recommended Practice

- Short statement on frontpage stating what is expected from the addressed recipients, for example:
  - Please send comments before february 29, this document will be reviewed on that date
  - This document is authorized, changes are only applied via a change request
- See Granularity of Documentation [3] for guidelines for modularization and contents

The example defines a minimum mandatory set. No layout guideline is given except the fact that a subset of the meta information is mandatory on every page.

This illustrates a very important aspect of templates:

The procedure is mandatory, the template is only an enabling means, which means that anyone can make its own template as long as it fulfills the mandatory rules of the procedure.

4.8 Pitfalls

The most frequent pitfalls in the application of templates are:

- Author follows template instead of considering the purpose of the document.
- Template is too complex.
- There is an unmanageable number of variants.
- Mandatory use of templates results in:
  - no innovation of templates (= no learning)
  - no common sense in deployment
  - strong dependency on templates

A tendency exists to put a lot of information and intelligence in a template. For instance specialized Word templates, which prompt for the required fields. These kinds of templates are very vulnerable with respect to tools and environment. Changes in tools, environment or process play havoc with these nice looking templates. Good templates are, as good designs, simple. Simple templates are easily understood and easily modified, providing flexibility, room for innovation and room for common sense by customization to the problem.

In due time the amount of specialized templates grows. As in a normal design re-factoring is required to keep the overall set simple and consistent and hence maintainable.

The most common pitfall is to make the template mandatory instead of making the procedure mandatory. In other words the **how** is enforced instead of the **what**. This is the main cause of all the following pitfalls, such as no innovation, no common sense and a strong dependency on templates. The mandatory use of templates inhibits the innovation and common sense by individual users.

Recommendation: Enforce the procedure *(what)*, provide the template *(how)* as supporting means.

4.9 Why I hate templates

Personally I hate templates. My way of working is based on immediate visual recognition of objects such as documents. For instance searching for a document on a large chaotic desktop is based on the visual image in my memory which is compared to the visual look of the documents on the desktop. When receiving a document the visual look immediately classifies the document with respect to author, project and status.

The uniformity caused by templates dramatically degrades my recognition performance and worse false matches turn up frequently.

This problem can be countered by allowing "personification" of documents, for instance by adding personal icons, images fonts et cetera. So by making variation on purpose!

Several people have pointed out to me that I violate my own needs for visual recognition with all Gaudí articles. Obviously here is room for improvement!

4.10 Summary

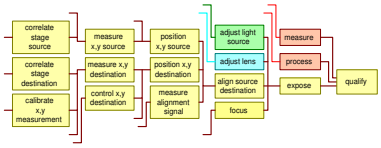
- Templates support (new) processes
- Use templates for layout and meta information support
- Do not use templates for documents structure or contents
- Stimulate evolution of templates, keep them alive
- Keep templates simple
- Standardize on **what** (process or procedure), not on **how** (tool and template)
- Provide (mandatory) guidelines and recommended practices
- Provide templates as a supportive choice, don't force people to use templates

4.11 Acknowledgements

Jürgen Müller identified the weak spots as usual, enabling an improved and more clear intermezzo. Discussion with Saar Muller helped to improve the terminology, such as guidelines and rules. The sharp eyes of Jaap v.d. Heijden helped to improve the figures.

Chapter 5

System Integration How-To



5.1 Introduction

Quality problems and delays are one of the symptoms of the troublesome relation between software and system The integration of software and hardware is in many organizations taking place when both hardware and software are nearly finished. Organizational boundaries propagate into the schedule causing too late integration of crucial technologies. Systems architects have to ensure that software-hardware integration starts very early.

System Integration is one of the activities of the Product Creation Process. The Product Creation Process starts with a set of product needs and ideas, and results in a system that:

- fits in customer's needs and context
- can be ordered, manufactured, installed, maintained, serviced, disposed
- fits the business needs

During Product Creation many activities are performed, such as: feasibility studies, requirements capturing, design, engineering, contracting suppliers, verification, testing, et cetera. Decomposition is an universal method used in organization, documentation and design. Decomposition enables the distribution of work in a concurrent fashion. The complement of decomposition is integration. Every activity that has been decomposed in smaller steps will have to be integrated again to obtain the single desired outcome.

Integration is an ongoing flow of activities during the entire product creation. The nature of integration activities, however, shifts over time. Early in the project technologies or components are integrated, while at the end of the project the entire system is built and verified. In formal process descriptions[1] the description of product integration is mostly limited to the very last phase of the total integration flow, with a focus on the administrative and process aspects. We use the term integration in the broader meaning of all activities where decomposed parts are brought together.

In practice projects hit many problems that are caused by decomposition steps. Whenever an activity is decomposed the decomposed activities normally run well, however crosscutting functionality and qualities suffer from the decomposition. Lack of ownership, lack of attention, and lack of communication across organizational boundaries are root causes for these problems. The counter measure for these problems is to have continuous attention for the integration.

5.1.1 Goal of Integration

The goal of integration is find unforeseen problems as early as possible, in order to solve these problems in time. Integration plays a major role in risk reduction. The word unforeseen indicates the main challenge of integration: How to find problems when you don't know that they exist at all?

Problems can be unforeseen because the knowledge of the creation team is limited. May be nobody on earth did have the knowledge to foresee such a problem, simply because the creation process enters new areas of knowledge. Problems can also be unforeseen due to invalid assumptions. For instance, many assumptions are being made early in the design to cope with many uncertainties. The limited intellectual capabilities of us, humans, limit also the degree in which we can oversee all consequences of uncertainties and of assumptions we make. A common source of unforeseen problems is interference between functions or components. For example, two software functions running on the same processor may perform well individually, but running concurrently may be way too slow, due to cache pollution or memory trashing.

5.1.2 Product Integration as part of Product Creation Process

The Product Creation Process (PCP) is often prescribed as a sequence of phases with increasing level of realization and decreasing level of risk. This is a useful high level mental model, however one should realize that most activities have much more overlap in the current dynamic world. The pure waterfall model, where requirements, design, integration and test are sequential phases, is not practical anymore. Much more practical is an approach with a shifting emphasis as shown

[1] for example NASA Procedural Requirements (NPR)

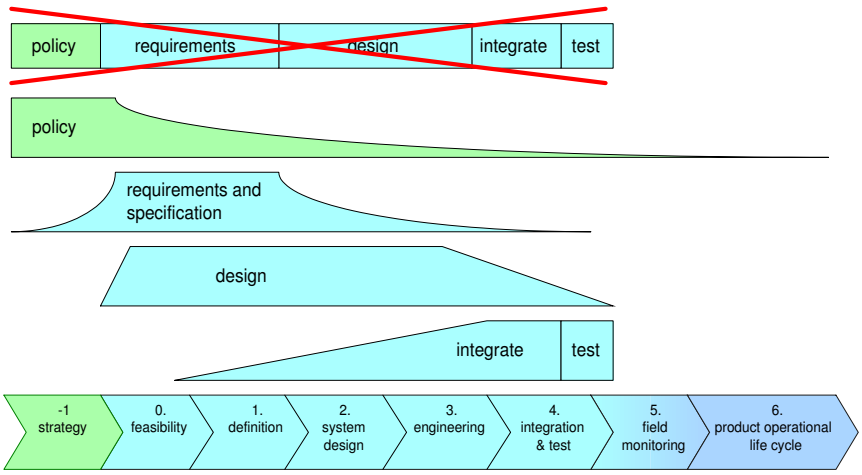


Figure 5.1: Typical Product Creation Process and the concurrency of engineering activities.

in Figure 5.1. A comparable approach is Rational Unified Process (RUP), see [5] and for integration [6]. Note especially the long ramp-up of the integration, the focus of this chapter.

5.1.3 Integration in Relation to Testing

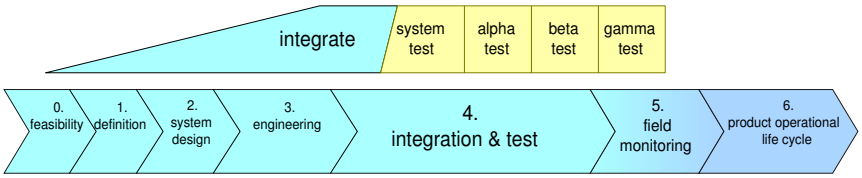


Figure 5.2: Zooming in on Integration and Tests

Integration and testing is often used as identical activities. However, the two activities are related and completely different at the same time.

Figure 5.2 zooms in on the integration and test activities. Integration is the activity where we try to find the unknowns and where we resolve the uncertainties. Testing is an activity where we operate a (part of a) system in a predefined manner and verify the behavior of the system. A test passes if the result fits the specified behavior and performance and otherwise it fails. Before integration starts testing is applied at component level. During integration many tests may be applied as part of the integration. These tests during integration are applied to find these unknowns and to resolve the uncertainties. When the milestone is passed that the system is perceived to be ready, then the systems engineers will run an entire system level

test suite. Normally, this run still reveals unknowns and problems. The system test verifies both the external specification, as well as the internal design. When sufficient stability of the system test is achieved a different working attitude is taken: from problem solving to verification and finishing. The alpha test starts with a hard milestone and is also finished at a well-defined moment in time. The alpha test is the formal test performed by the product creation team itself, where the specification is verified. The beta test is also a well-defined time-limited formal test, performed by the "consuming" internal stakeholders: marketing, application, production, logistics and service. The beta test also verifies the specification, but the testers have not been involved in product creation. These testers are not blinded by their a priori know-how. Finally the external stakeholders, such as actual users, test the product. Normally, problems are still found and solved during these tests, violating the assumption that the system is stable and unchanged during testing. In fact, these alpha, beta, and gamma testers hit problems that should have been found during integration. We will focus the rest of this chapter on integration, with the main purpose to reduce risks in the testing phase by identifying (potential) problems as early as possible.

5.2 What, How, When and Who of Integration

By necessity the integration of a system starts bottom-up with testing individual components in a provisional component context. The purpose of the bottom-up steps is to find problems in a sufficiently small scope; the scope must be small enough to allow diagnosis in case of failure. If we bring thousands of components together into a system, then this system will fail for certain. But it is nearly impossible to find the sources of this failure, due to the multitude of unknowns, uncertainties, and ill-functioning parts.

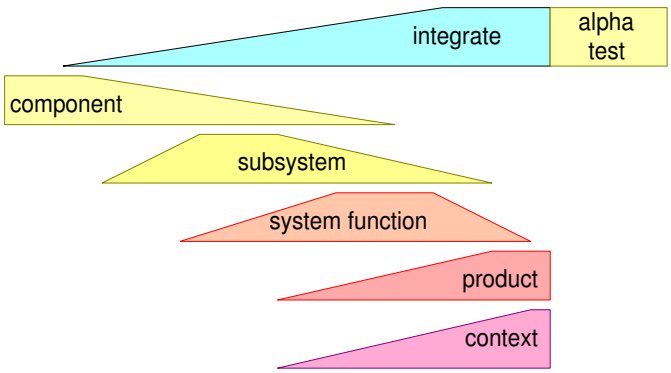


Figure 5.3: Integration takes place in a bottom-up fashion, with large overlaps between the integration levels.

The focus of the integration activity is shifting during the integration phase. Figure 5.3 shows the bottom-up levels of integration over time. Essential for integration is that the higher levels of integration start already, when the lower levels of integration are not yet finished. The different levels of integration are therefore overlapping. Early during integration the focus is on functionality and behavior of components and subsystems. Then the focus is shifting to system level functionality: do the subsystems together operate in the right way? The last step in integration is to focus on the system qualities, such as performance and reliability.

Figure 5.4: During integration a transition takes place from using previous systems and partial systems to the new system configuration.

The integrator tries to integrate subsystems or functions as early as possible with the purpose of finding unforeseen problems as early as possible. This means that integration already takes place, while most of the new components, subsystems, and functions are still being developed. Normally partial systems or modified existing systems are used in the early phases of integration as substitute of the not yet available parts. Figure 5.4 shows this transition from using partial and existing subsystems to systems based on new developed parts.

The unavailability of subsystems or the lack of stability of new subsystems forces the integrator to use alternatives. Figure 5.5 shows a classification of alternatives. Simple stubs in a virtual environment up to real physical subsystems in a physical environment can be used. In practice multiple alternatives are combined. As function of time the integration shifts from the use of stubs and a virtual environment to as close as possible to the final physical reality.

The challenge for the project team is to determine what intermediate integration configurations are beneficial. Every additional configuration adds costs: creation costs as well as costs to keep it up-to-date and running. An even more difficult

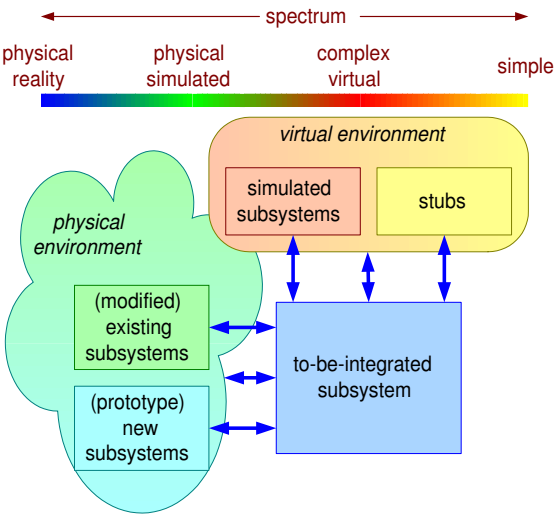


Figure 5.5: Alternatives to integrate a subsystem early in the project.

conflict is that the same critical resources, dynamic positioning experts for instance, are needed for the different configurations. Do we focus completely on the final product, or do we invest in intermediate steps? Last but not least is the configuration management problem that is created with all integration configurations. When hundreds or thousands of engineers are working on a product then most of them are in fact busy with changing implementations. Strict change procedures for integration configurations may reduce the management problem, but this conflicts often with the troubleshooting needs during integration.

Crucial questions in determining what intermediate configurations to create are:

- How critical or sensitive is the subsystem or function to be integrated?
- What are the aspects that are sufficiently close to final operation so that the feedback from the configuration makes sense?
- How much needs to be invested in this intermediate step? Special attention is required for critical resources.
- Can we formulate the goal of this integration system in such a way that it guides the configuration management problem?

Based on these considerations we propose a stepwise integration approach as shown in Figure 5.6. The first step is to determine a limited set of the most critical system performance parameters, such as image quality, productivity or power consumption. These system performance parameters are the outcome of

1	Determine most critical system performance parameters.
2	Identify subsystems and functions involved in these parameters.
3	Work towards integration configurations along these chains of subsystems and functions.
4	Show system performance parameter as early as possible; start with showing "typical" system performance.
5	Show "worst-case" and "boundary" system performance.
6	Rework manual integration tests in steps into automated regression tests.
7	Monitor regression results with human-driven analysis.
8	Integrate the chains: show system performance of different parameters simultaneously on the same system.

Figure 5.6: Stepwise integration approach

a complicated interaction of system functions and subsystems; we call the set of functions and subsystems that result in a system parameter a chain. We start to define partial system configurations as integration vehicles once we have identified critical chains. The critical chains serve as guidance for the integration process.

We strongly recommend focusing on showing the critical system performance parameters as early as possible. In the beginning the focus is on “typical” performance. Once the system gets somewhat more stable and predictable, then we get room to also study “worst-case” and “boundary” performance.

It is important to monitor the system performance regularly, since many engineers are still changing many parts of the total system. The early integration tests are manual tests, because the system circumstances are still very premature and because integrators have to be responsive to many unexpected problems. In due time the chain and the surrounding system gets more stable, allowing automation of tests. We can migrate the early manual integration steps into automated regression test. The results of regularly performed regression tests must be monitored and analyzed by system engineers. This analysis does not focus on pass or fail, but rather looks for trends, unexplained discontinuities, or variations.

Later during integration we have to integrate the chains themselves and to show the simultaneous performance of the critical performance parameters.

The approach described above requires quite some logistics support. The project leader will therefore make integration schedules in close cooperation with the system engineers. Integration schedules have two conflicting attributes:

Predictability and stability to ensure timely availability of resources

Flexibility and agility to cope with the inherent uncertainties and unknowns.

The starting point to create a schedule is to determine a specific and detailed integration order of components and functions. The integration order is designed such that the desired critical system performance parameter can be measured as early as possible.

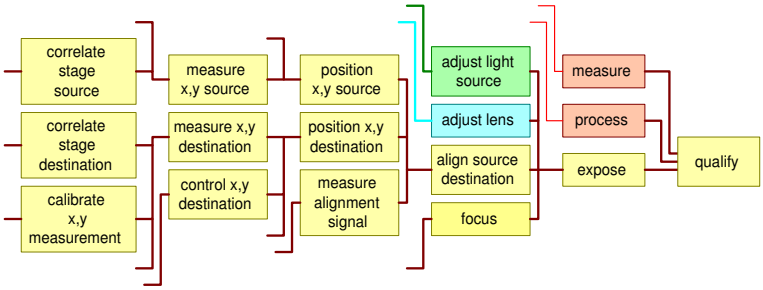


Figure 5.7: Example of small part of the order of functions required for the image quality system performance parameter of a wafer stepper.

Figure 5.7 shows an example of a specific order of functions required to determine the image quality system performance parameter of a wafer stepper. Such a diagram starts often at the right hand side: what is the desired output parameter to be achieved? Next the question "What is needed to achieve this output?"' is asked recursively. This very partial diagram is still highly simplified. In reality many of these functions have multiple dependencies.

Worse is that often circular dependencies exist, for instance in order to align source and destination we need to be in focus, while in order to find the focal plane we need to be aligned. These dependencies are known during design time and already solved at that moment. For example, a frequently used design pattern is a stepwise refined: coarse and fine alignment, and coarse and fine focusing. The creation of a detailed integration schedule provides worthwhile inputs for the design itself. Making the integration schedule specific forces the design team to analyze the design from integration perspective and often results in the discovery of many (unresolved) implicit assumptions.

The existence of this integration schedule must be taken with a grain of salt. It has a large value for the design and for understanding the integration. Unfortunately, the integration process itself turns out to be poorly predictable: it is an ongoing set of crises and disruptive events, such as late deliveries, breaking down components, non-functioning configurations, missing expertise, wrong tolerances, interfering artifacts, et cetera. Crucial to the integration process are capabilities to improvise and to troubleshoot.

The integration schedule is a rather volatile, and dynamic entity. It does not make sense to formalize the integration heavily, neither to keep it updated in all

details. Formalization and extensive updating takes a lot of effort with little or no benefits. The recommended approach is to use the original integration schedule as kind of reference and to use short cyclic planning steps to guide the integration process. Typical meeting frequency during integration is once per day. Every meeting results and problems, required activities and resources, and short-term schedule are discussed.

During integration many project team members are involved with different roles and responsibilities:

- Project leader
- System architect/engineer/integrator
- System tester
- Logistics and administrative support personnel
- Engineers
- Machine owner

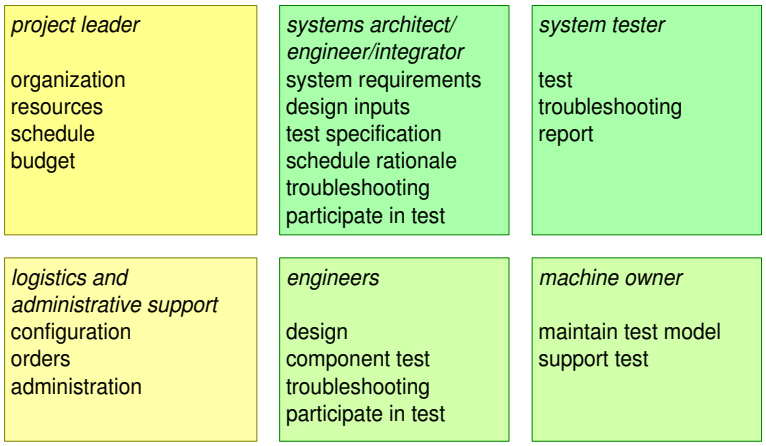


Figure 5.8: Roles and responsibilities during the integration process.

Figure 5.8 shows these roles in relation to their responsibilities. Note that the actual names of these roles depend on the organization, we will use these generic labels in this chapter.

The *project leader* is the organizer who takes care of managing resources, schedule and budget. Based on inputs from the system engineer the project leader will claim and chase the required resources. The project leader facilitates the integration process. This contribution is critical for the project timing.

The *system architect*, *systems engineer* and *system integrator* role is in fact a spectrum of roles that can be performed by one or more persons, depending on their capabilities. A good system architect is sometimes a bad system integrator and vice versa[2]. This role is driven by content, relating critical system performance parameters to design and to test. In this role the rationale of the integration schedule is determined and the initial integration schedule is a joint effort of project leader and systems engineer. The integral perspective of this role results in a natural contribution to the troubleshooting.

The *system tester* is the practical person actually performing most of the tests. During the integration phase lots of time of the system tester is spent in troubleshooting, often of trivial problems. More difficult problems are escalated to engineers or system integrator. The system tester documents test results in reports.

The *machine owner* is responsible for maintaining a working up-to-date test model. In practice this is a quite challenging job, because many engineers are busy with making updates and performing local tests, while system integrator and system tester need undisturbed access to a stable test model. We have observed that explicit ownership of one test model by one machine owner increases the test model stability significantly. Organizations without such role lose a lot of time due to test model configuration problems.

Engineers deliver locally tested and working components, functions or subsystems. However, the responsibility of the engineers continues into the integration effort. Engineers participate in integration tests and help in troubleshooting.

The project team is supported by all kinds of support personnel. For integration the *logistics and administrative support* is crucial. They perform configuration management of test models as well as the products to be manufactured. Note that integration problems may induce changes in the formalized product documentation and the logistics of the final manufacturing, which can have significant financial consequences due to the concurrency of development and preparation of production. The logistics support people also have to manage and organize unexpected but critical orders for parts of test models.

5.3 Configuration Management

Configuration management and integration are intimately related as discussed in the previous sections. We should realize that configuration management plays a role in many processes. Figure 5.9 shows a simplified process decomposition of those processes that are related to configuration management.

Basically, the internal *Customer Oriented* and *Product Creation* processes are

[2]Critical characteristics for architects are the balance theoretical versus hands-on, conceptual versus implementation, creative and diverging versus result driven and converging. During integration the emphasis must be on hands-on, implementation, and result driven an converging.

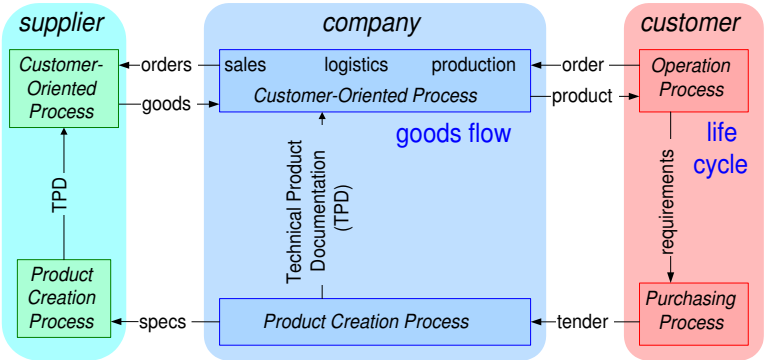


Figure 5.9: Simplified Process diagram that shows processes that are relevant from configuration management perspective.

linked to the related supplier and customer processes. There are two main flows where configuration management plays a role:

- Creation flow, from customer requirements to component specifications to technical product documentation to be used in the other flow.
- Goods flow, a repeating set of processes where orders are fulfilled by a logistics and production chain.

In principle the creation flow is a one-time project activity. This flow may be repeated to create successor products, but this is a new instantiation of this flow. The goods flow is a continuous flow with life cycle considerations. The final product as used operationally by customers also has its own life cycle.

Many entities have changing configurations and therefore need configuration management. Figure 5.10 shows the same process decomposition as Figure 5.9, but now annotated by entities under configuration management. Two classes of configuration management entities exist: information and physical items. The information entities are normally managed by procedures and computer based tools. However for physical entities the challenge is to maintain consistency between the actual physical item and the data in the configuration management administration. Especially during the hectic period of integration the administration sometimes differs from the physical reality, causing many nasty problems. Sometimes more effort in processes helps, however, sometimes more effort in processes results in more latency and more work-around behavior; unfortunately, there is no silver bullet for configuration management processes.

The main configuration management entities during integration are the test models. Changes in test models may have to propagate to other entities, such as specifications, technical product documentation, and, due to concurrency, also to components and products in the goods flow processes.

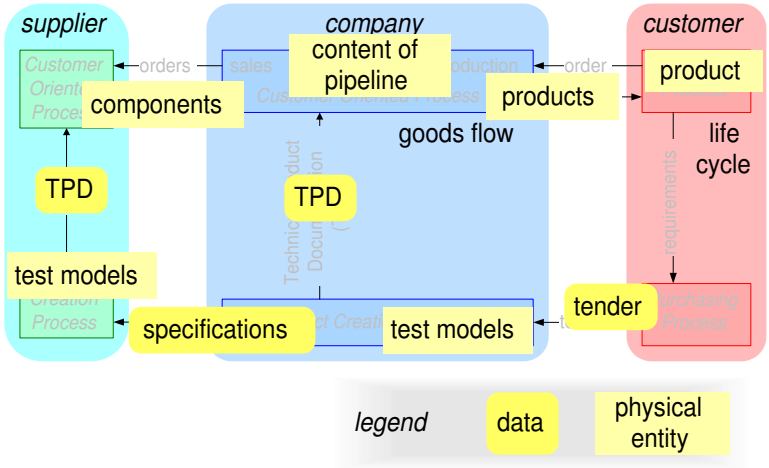


Figure 5.10: The simplified process diagram annotated with entities that are under configuration management.

One particular area of attention is the synchronization of components, subsystems and test models. All these entities exist and change concurrently. A certain pull to use latest versions is caused by the fact that most problems are solved in the latest version. However, integrators and testers need a certain stability of a test model. This makes integrators and testers hesitant to take over changes. One should realize that only a limited amount of test models exist, while all these engineers create thousands of changes. On top of this problem comes a logistics problem: from change idea to availability of changed component or function may take days or weeks. Sometimes one single provisionally changed component is available early.

One way of coping with the diversity of test model configurations is to clearly formulate the integration goals of the different test models. Note that these integration goals may change over time, according to Figure 5.3.

5.4 Typical Order of Integration Problems Occurring in Real Life

Experience in many integration phases resulted in the observation of a typical order when integration problems occur. This typical order is shown in Figure 5.11.

Typically none of these problems should occur, but despite mature processes all of them occur in practice. The failure to build the system at all is often caused by the use of implicit know-how. For example, a relatively addressed data file that resides on the engineers workbench, but that is not present in the independent test environment. As a side remark we observe the tension between using networked test models. Network connections shorten software change cycles and help in

1. The (sub)system does not build.
2. The (sub)system does not function.
3. Interface errors.
4. The (sub)system is too slow.
5. Problems with the main performance parameter, such as image quality.
6. The (sub)system is not reliable.

Figure 5.11: Typical Order of Integration Problems

troubleshooting, however, at the same time the type of problems we discussed here may stay invisible.

The next phase in integration appears to be that individual components or functions work, but cease to function when they are combined. Again the source of the problem is often a violated implicit assumption. This might relate to the third problem, interface errors. The problem might be in the interface itself, for instance different interpretations of the interface specification may result in failures of the combination. Another type of problem in this category is again caused by implicit assumptions. For example, the implementation of the calling subsystem is based on assumed functionality of the called subsystem. It will be clear that different than assumed behavior of the called subsystem may cause problems for the caller. These types of problems are often not visible at interface specification level, because none of the subsystem designers realized that the behavior is relevant at interface level.

Once the system gets operational functionally, then the non-functional system properties become observable. The first problem that is hit in this phase by integrators is often system performance in terms of speed or throughput. Individual functions and components in isolation perform well, but when all functionality is running concurrently sharing the computing resources then the actual performance can be measured. The mismatch of expected and actual performance is not only caused by concurrency and sharing, but also by the increased load of more realistic test data. On top of these problems non-linear effects appear when the system resources are more heavily loaded, worsening overall performance even more. After some redesigns the performance problems tend to be solved, although continuous monitoring is recommended. Performance tends to degrade further during integration, due to added functionality and solutions for other integration problems.

When the system is both functional and well performing, then the core functionality, the main purpose of the product, is tested extensively. In this phase the application experts are closely involved in integration. These application experts use the system differently and look differently at the results. Problems in the critical system functionality are discovered in this phase. Although these problems were

already present in the earlier phases, they stayed invisible due to the dominance of the other integration problems and due to the different perspectives of technical testers and application experts.

During the last integration phase the system gets used more and more intensively. The result is that less robust parts of the design are exercised more causing system crashes. A common complaint in this phase is that the system is unreliable and unstable. Part of this problem is caused by the continuous influx of design changes triggered by the earlier design phases, every change also triggers new problems.

5.5 Acknowledgements

Dinesh Verma stimulated me to write this paper. Roland Mathijssen provided feedback and citations.

Bibliography

[1] Thomas Gilb and Dorothy Graham. *Software Inspection*. Addison-Wesley, 1993.

[2] James Morgan. Applying lean principles to product development. http://www.sae.org/manufacturing/lean/column/leanfeb02.htm, 2010.

[3] Gerrit Muller. Granularity of documentation. http://www.gaudisite.nl/DocumentationGranularityPaper.pdf, 1999.

[4] Gerrit Muller. The system architecture homepage. http://www.gaudisite.nl/index.html, 1999.

[5] Wikipedia. Rational unified process (rup). http://en.wikipedia.org/wiki/Rational_Unified_Process, 2006.

[6] Wikipedia. Rup test discipline. http://en.wikipedia.org/wiki/Rational_Unified_Process#Test_Discipline, 2006.

History

Version: 1.2, date: September 10, 2010 changed by: Gerrit Muller

- Added "Lean and A3"

Version: 1.1, date: March 29, 2004 changed by: Gerrit Muller

- Added "Light Weight Review Process"

Version: 1.0, date: March 25, 2004 changed by: Gerrit Muller

- created reader