

Additions to "Systems Design and Engineering"

SESAME

Research group Mechatronics

Enschede, June 7, 2022, version v1.2.2

Additions to "Systems Design and Engineering"

Authors: Victor Sluiter and Mark Reiling

Table of Contents

Chapter 1 Introduction to the addition to Systems Design and Engineering	6
1.1 A word about formality	6
1.1.1 References	7
Chapter 2 The Systems Engineering Process	8
2.1 Addition to 2.3, A practical implementation of SE	8
2.1.1 Agile development	8
2.1.2 References	9
Chapter 3 Systems Thinking Tracks	11
3.1 Addition to 3.13, Risk Thinking	11
3.1.1 Pre-Mortem technique	11
3.1.2 NPR 5326:2019	11
3.1.2.1 References	12
3.2 Addition to 3.10, Organizational Thinking	12
3.2.1 Capability Maturity Model (CMM)	12
3.2.1.1 Why is this important to Systems Engineering?	14
3.2.2 References	16
Chapter 4 Systems Design Tools	17
4.1 Addition to 4.7 and 4.13 ; Architecture Decision Records	17
4.1.1 What is an ADR?	17
4.1.2 Using Architecture decisions for reverse architecting and reviewing	19
4.1.3 Related subjects	19
4.1.3.1 Technical Debt	19
4.1.3.2 ARC42	20
4.1.4 References	20
4.2 Addition to 4.9, FunKey Architecting and ValueFirst	21
4.2.1 Explanation of ValueFirst / Planguage	21
4.2.1.1 Another view on Functions... ..	22
4.2.1.2 ... And how values "trickle down" to solutions	22
4.2.1.3 ... Leads to comparison matrices!	23
4.2.2 References	24
4.3 Addition to 4.14, Simulation	25
4.3.1 Other forms of simulation	25
4.3.2 Digital Twin	25
4.3.3 References	26
4.4 Model Based Systems Engineering	26

4.4.1	MBSE in short	26
4.4.2	Benefits and caveats	27
4.4.2.1	MBSE for SME	28
4.4.3	References	29
Chapter 5	Additional subjects	30
5.1	Requirements Languages	30
5.1.1	Planguage	30
5.1.1.1	Explanation of ValueFirst / Planguage	30
5.1.1.2	Examples of research group	31
5.1.1.3	Cheat sheet	33
5.1.1.4	References	34
5.1.2	EARS.....	34
5.1.2.1	Example from usage at the research group	35
5.1.2.2	Cheatsheet from Aalto university.....	36
5.1.2.3	References	37
5.2	Incremental Delivery of hardware.....	37
5.2.1	References	38
5.2.2	EVO	38
5.2.2.1	How to “do” EVO?	38
5.2.2.2	References	41
5.2.3	Rapid Learning Cycles	42
5.2.3.1	Way of thinking	42
5.2.3.2	References	44
5.3	Product Line Engineering	44
5.4	A great additional resource.....	45

List of Figures

Figure 1	Section from [2], p.36	7
Figure 2	Relation between SMEs and their SE tools, from a study on Australian SMEs.	7
Figure 3	From [1]	9
Figure 4	Table of contents of NPR 5326.....	12
Figure 5	Example Architecture Decision Record	18
Figure 6	Overview of ARC42 documentation template.....	20
Figure 7	Article from Bits & Chips Magazine	21
Figure 8	How performance qualities relate to functions. Source: [3]	22
Figure 9	Screenshot of ValPlan	23

Figure 10 "Impact Estimation Table" recently re-dubbed as "Value Estimation Table". This table uses the quantified values (here "objectives") and costs to make a Performance to Cost Estimation. Source: [3]	24
Figure 11 Examples of Tools, Modelling Languages and Methods for MBSE. Source: [2]	27
Figure 12 When fully embedded in the organization, this integration is possible using MBSE. Source: [2]	28
Figure 13 How performance qualities relate to functions. Source: [1].....	31
Figure 14 Used with permission by Kai Gilb, www.gilb.com.....	33
Figure 15 EARS language rules. Source: [3].....	35
Figure 16 EVO cycles are small waterfalls. Source: [4].....	38
Figure 17 Cycles in Evo. Source:[4]	39
Figure 18 Some delivery needs to be prepared in earlier cycles. This is also planned in EVO.	40
Figure 19 Example Impact Estimation Table. Source: [3].....	41
Figure 20 Still from "When Agile Gets Physical" talk. See[2].....	42
Figure 21 Elements of RLC. Source: [1].....	43
Figure 22 Not every decision is a key decision. Source:[1].....	43
Figure 23 Learning Cycles Plan. KG=Knowledge Gap, KD=Key Decision. Source: [1].....	43
Figure 24 Shortening development by reusing knowledge. Source: [3]	44

List of Tables

No tables included in this document

Version	Description	Saved by	Saved on	Status
v1.2.2	Reworked part on CMMI, added authors	Victor Sluiter	Jun 7, 2022 9:42 AM	APPROVED
v1.2.1	Minor update, removed some notes.	Victor Sluiter	Dec 24, 2021 3:12 PM	APPROVED
v1.2.0	Added stub about Product Line Engineering.	Victor Sluiter	Dec 24, 2021 3:11 PM	APPROVED
v1.1.0	Added EARS and Planguage sections, added EVO and RLC	Victor Sluiter	Dec 23, 2021 12:05 PM	APPROVED
v1.0.0	Temporary version to share with partners	Victor Sluiter	Dec 3, 2021 11:02 AM	APPROVED
v0.0.6	Added Agile view	Victor Sluiter	Dec 2, 2021 10:11 AM	APPROVED
v0.0.5	Added CMM, added MBSE	Victor Sluiter	Nov 30, 2021 10:06 PM	APPROVED
v0.0.4	Added ValueFirst / Funkey and skeleton for other subjects.	Victor Sluiter	Nov 29, 2021 11:02 PM	APPROVED
v0.0.3	Added Simulation, and in added formality introduction	Victor Sluiter	Nov 12, 2021 12:17 PM	IN PROGRESS
v0.0.2	Added ADR	Victor Sluiter	Nov 10, 2021 10:08 AM	IN PROGRESS
v0.0.1	Initial version	Victor Sluiter	Nov 3, 2021 2:01 PM	IN PROGRESS

Chapter 1 Introduction to the addition to Systems Design and Engineering

In the SESAME project, we're investigating what Systems Engineering (SE) methods and solutions could help the Small to Medium Enterprises (SMEs) to improve their efficiency and quality of design and production. One of the deliverables that the SESAME project would generate would be an overview of current "best practices" in SE. This is quite a daunting task, and luckily a lot of prior work has been done by G. Maarten Bonnema, Karel Th. Veenvliet and Jan F. Broenink in their book "Systems Design and Engineering"[1].

We decided to take that book as a basis, and describe tools and techniques that seem interesting for the companies involved in our project in the "framework" of the book by Bonnema et. al. This means we add information we think is useful to The Systems Engineering Process, the Systems Thinking Tracks and Systems Design Tools. You can read this information "on its own", but it is recommended to keep it next to the book, and read our additions at the corresponding sections of the book.

The last chapter can be read as an Appendix. It takes some subjects that do not neatly fit into the book, but that we thought were too interesting for this project to leave to our own!

1.1 A word about formality

While researching and thinking about methods and insights that might help the current group of companies contributing to SESAME, I found it hard to determine what the common feature was that is needed for these companies, but also for our own work in the research group. During the discussions we had a lot of overlap on insights on workflows, roles, and ways of communication and documentation.

It was not that the tools presented should be *simple*. An A3 Architecture Overview can be quite complex. The designs made by Hencon, Hollander Techniek and Riwo are not *simple*. Also, I do not feel a grudge against documentation. On the other hand, none of the partners feel at home making strict UML graphs, or using strict architecting techniques and maintain that knowledge among their employees. That would feel like a waste of time.

While reading the dissertation of Sandra Schröder [2] I found a table with "Formalism" on the horizontal axis. I think this is something where we can see that where currently most partners are at "informal", we'd like to step up to "semi-formal" to improve shareability of designs and thinking tracks but keep a lot of liberty in implementation that is no longer present in a strict "formal" way of working.

Notation	textual	Prose text	arc42 templates Architecture Decision Records Controlled Natural Language	Architecture Description Languages Domain Specific Languages Controlled Natural Language
	graphical	Boxes-and-Lines Diagrams	Unified Modeling Language	Architecture Description Languages Domain Specific Languages
		informal	semi-formal	formal

Formality

Figure 2.2.: Examples of modeling languages and their classification according to the dimensions “formality” and “notation”.

Figure 1 Section from [2], p.36

A similar “view on relationships between SMEs and SE tools can be found in [3]. I think for R&D departments - even in the context of large enterprises - we’re mostly looking at the “attitude” of Small SMEs:

Table 3 – Preliminary view of relationships between SMEs and SE tools

	Small SMEs (1-50 fulltime staff)	Large SMEs (1-250 fulltime staff)
Choice of tools	Based on their type of business	Dictated by overseas parent companies
Process	Lean and flexible culture with minimal formal controls	Stringent with culture established by overseas parent companies
Type of tools	Inexpensive, FOSS or IDS	COTS
Perspective on current tools	Preferring simple, more affordable tools Preferring more education or guidelines on current tools than on obtaining new tools	Being content with current COTS tools Satisfied with the technical support and training from parents companies

Figure 2 Relation between SMEs and their SE tools, from a study on Australian SMEs.

I hope I struck the right chord here, if so, I think you will find the additions presented here interesting for you!

1.1.1 References

1. Bonnema, G. Maarten, Karel Th Veenvliet, and Jan F. Broenink. *Systems Design and Engineering: Facilitating Multidisciplinary Development Projects*. Boca Raton: CRC Press, Taylor & Francis Group, 2016.
2. Schröder, Sandra. “Ontology-Based Architecture Enforcement: Defining and Enforcing Software Architecture as a Concept Language Using Ontologies and a Controlled Natural Language,” November 2020. <https://ediss.sub.uni-hamburg.de/handle/ediss/8671>.
3. Tran, Xuan-Linh, Timothy Ferris, Thomas V. Huynh, and Shruga Shoval. “10.2.2 Research on a Framework for Systems Engineering Tools for Australian Small and Medium Enterprises.” *INCOSE International Symposium* 18, no. 1 (June 2008): 1104–19. <https://doi.org/10.1002/j.2334-5837.2008.tb00866.x>.

Chapter 2 The Systems Engineering Process

2.1 Addition to 2.3, A practical implementation of SE

2.1.1 Agile development

In 2.3.5, the Vee model is presented. This model is well known for infrastructure projects and large scale projects. First note that there is no standardized Vee model. Different companies use different implementations, and different processes to “go through the Vee”. The most important aspect is that thinking about the tests should be done while drawing up the requirements.

One of the drawbacks of how the Vee model is regularly used is that first most of the requirements are “set in stone”, hopefully together with tests in the right side of the Vee. When during implementation and testing inconsistencies in requirements are found, or requirements are changed, the whole Vee has to be revisited again. Do the tests still fit the requirements? Are the functional concepts / architectures still applicable to these new requirements? Do subsystem tests have to be redefined? Do these changes have implications for other subsystems that are developed? Although it is very **good** that these questions are asked, the amount of paperwork to update all documents can be daunting.

When doing innovative projects, the amount of changes during development can be quite large, also because some requirements are still not completely known. Originating from software development, another approach can be taken which is *agile* development, as opposed to the *rigid* structure that many Vee model implementations have.

The key to agile development is to have many iterative design changes, where parts of a product are developed in very short timeframes. These parts are then shown to the end customer, who can immediately react and change course of the project when needed. Of course it is harder to predict the duration of the project when doing it this way.

The most well-known method to do agile development is SCRUM. Although the goal is to be agile in the long run, SCRUM has very *strict* rules on the execution and scope of the project during the short incremental changes. In general, SCRUM is well suited for continuous development on software products, but not for innovative products with lots of (un)known unknowns and lead times for components[2].

Interesting side note: the term SCRUM was derived from the paper about “The New Product Development Game” that highlighted some “Corporate Rugby Scores” of companies that developed faster by letting engineers talk to customers and other engineers, and develop multiple subsystems simultaneously. This paper was from 1986 and mostly dealt with hardware delivery; cars, printers, copiers, power tools! <https://hbr.org/1986/01/the-new-new-product-development-game>

The effect is that currently the most common way of work is to do a Vee model approach, but let the software development use an agile method, mostly SCRUM. From *Managing Requirements Volatility While ‘Scrumming’ within the V-Model* [1]:

However, in the case of large projects (especially safety-critical and medical system development), it is necessary that requirement specifications are continuously reviewed at every level of the project, regardless of what development methodology is being followed. This is because regulatory and legal requirements have to be complied with at an overall system level, and is another reason why the V model cannot be completely removed – safety and regulatory standards must be rigorously adhered to, so that there is less work for the Scrum teams.

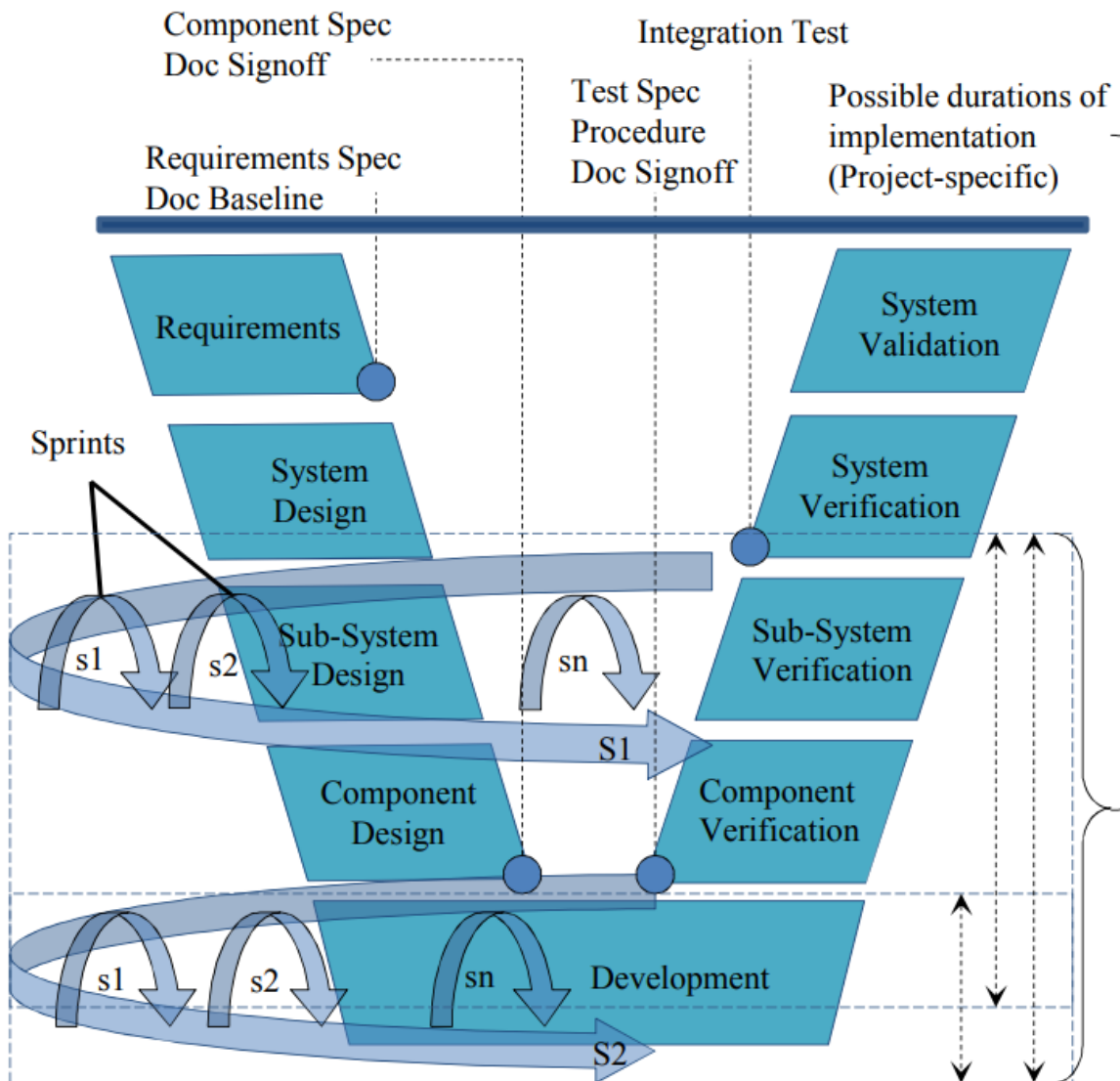


Fig. 1. Scrumming within the V-Model

Figure 3 From [1]

So can't we be more agile in delivering innovative products with hardware? Yes, we can, but we need a method that keeps the long term goals in mind: the customer wish and the complete product lifecycle, as described in the book. This was also highlighted in a very recent article (Nov 2021) in Mechatronica & Machinebouw: "*Maak sneller fouten*"¹[3]. Two examples of methods for this are described in *Incremental Delivery of hardware*(see page 37) .

2.1.2 References

1. Anitha, P.C., Deepti Savio, and V. S. Mani. "Managing Requirements Volatility While 'Scrumming' within the V-Model." In *2013 3rd International Workshop on Empirical Requirements Engineering (EmpiRE)*, 17–23, 2013. <https://doi.org/10.1109/EmpiRE.2013.6615211> .
2. High Velocity Innovation. "Agile for Hardware: When Agile Gets Physical," April 19, 2021. <https://highvelocityinnovation.com/agile-for-hardware-when-agile-gets-physical/> .

¹ <https://mechatronicamachinebouw.nl/artikel/maak-sneller-fouten/>

-
3. "Maak sneller fouten – Mechatronica&Machinebouw." Accessed December 2, 2021. <https://mechatronicamachinebouw.nl/artikel/maak-sneller-fouten/> .

Chapter 3 Systems Thinking Tracks

3.1 Addition to 3.13, Risk Thinking

3.1.1 Pre-Mortem technique

To aid in risk thinking, there's a powerful "trick" that was described in the book *Meltdown*[1], which in turn got it from an article in *Harvard Business Review*[2]. The trick is to do a premortem analysis, which generates different failure scenarios than simply asking "what could go wrong". The technique can be used by yourself, for yourself, but also when working in a team.

When projects fail, it's normal to do a "postmortem", i.e. check why the project "died". The premortem analysis is done before a project starts, and is done by telling the discussion members to imagine that x years from now the project failed miserably. The team members are then asked to come up with reasons *why* it failed.

Using this prospective hindsight generates other scenarios than asking "what could go wrong?" before starting the project. Reasons given are clearer story lines leading up to the failure of the project or design[3]. This works because the human mind is better at explaining certain outcomes than explaining uncertain outputs. The result of a premortem can be a more diverse list of possible things that can go wrong, which the systems engineer can keep in mind.

3.1.2 NPR 5326:2019

The Dutch norms committee NEN has published a "Nederlandse Praktijk Richtlijn" ( Dutch Practice Guidelines) for "Quality assurance of custom software development and maintenance". This guideline, NPR 5326 was made to prevent software projects requested by the government run out of control. The risks and control measures mentioned in this free and openly accessible document are usable for any risky technology development. Please see <https://www.nen.nl/npr-5326-2019-nl-262885> for more detail. In the available download you can even find an Excel sheet to scan your own project for possible risks.

Contents

Foreword	4
1 Scope	5
2 References.....	5
3 Terms and definitions.....	6
4 Abbreviations	15
5 Explanatory note on terms.....	15
5.1 The term risk.....	15
5.2 The term control.....	16
6 Risks	17
6.1 General.....	17
6.2 Product-related risks.....	17
6.3 Project-related risks	18
7 Controls.....	23
7.1 General.....	23
7.2 Project-related controls.....	23
7.3 Organization-related controls.....	33
Annex A Overview of risks and controls	36
Annex B Assessment tool	38
Bibliography.....	40

Figure 4 Table of contents of NPR 5326

3.1.2.1 References

1. CLEARFIELD, CHRISTOPHER. TILCSIK, ANDRAS. *MELTDOWN: Why Our Systems Fail and What We Can Do about It*. Place of publication not identified: ATLANTIC Books, 2019.
2. Klein, Gary. "Performing a Project Premortem." *Harvard Business Review*, September 1, 2007. <https://hbr.org/2007/09/performing-a-project-premortem> .
3. Mitchell, Deborah, J. Russo, and Nancy Pennington. "Back to the Future: Temporal Perspective in the Explanation of Events." *Journal of Behavioral Decision Making* 2 (January 1, 1989): 25–38. <https://doi.org/10.1002/bdm.3960020103> .

3.2 Addition to 3.10, Organizational Thinking

3.2.1 Capability Maturity Model (CMM)

In 1991, the Software Engineering Institute (SEI) of Carnegie Mellon University made a framework to help government agencies with software projects going out of control (note: this is also the source of the Dutch NPR in [Addition to 3.13, Risk Thinking](#)(see [page 11](#))). SEI came up with the Capability Maturity Model for Software , updated by Mark Paulk in 1993 [1].

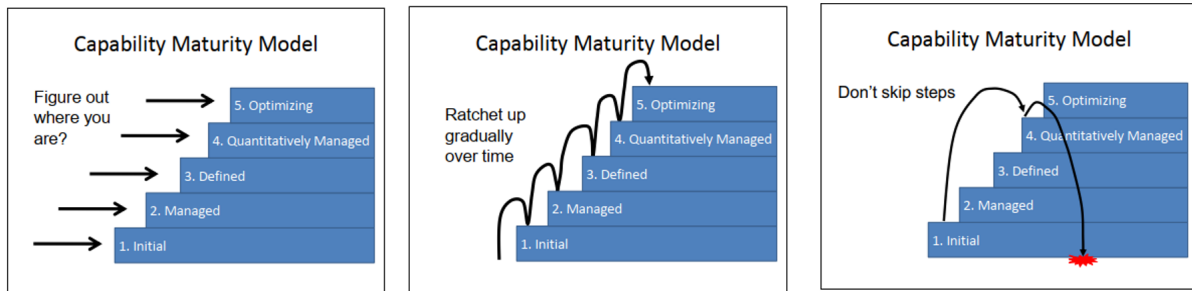
The solution was not a strict set of rules, but a characterization to see what organizations had which maturity in organizing themselves. The very nice part about it, is that it gives a framework that is more widely applicable than only software, or even only engineering.

It was observed that many projects don't deliver quality or timeliness that was expected, even if very strict processes were promised. On the other hand, some smaller vendors would deliver excellent result without any disciplined engineering. Repeating those results relied heavily on having the same

individuals doing the sequential projects. This is not sustainable, nor does that give possibilities for continuous improvement.

CMM puts organizational processes in "maturity levels", where Level 1 is completely unmanaged, and Level 5 is completely quantitatively managed throughout the project. These levels are also named Initial, Repeatable, Defined, Managed, Optimizing. By assessing your organizational processes, you can try to optimize by advancing one of the processes one step, and then try to remain that level, very much like a lean Plan-Do-Check-Act cycle.

Michael Edson and Nik Honeysett have done a great job to explain how they used CMM in software projects in the Getty Museum and the Smithsonian institute [2]:



As a "quick scan" you can use the following chart, from [2] and [3]. The columns are the maturity levels, the rows are the way the maturity levels "act" on internal processes, people, integrating new technology and how process information is measured:

Implications of Advancing Through CMM Levels.					
	Level 1	Level 2	Level 3	Level 4	Level 5
Processes	Few stable processes exist or are used.	Documented and stable estimating, planning, and commitment processes are at the project level.	Integrated management and engineering processes are used across the organization.	Processes are quantitatively understood and stabilized.	Processes are continuously and systematically improved.
	"Just do it"	Problems are recognized and corrected as they occur.	Problems are anticipated and prevented, or their impacts are minimized.	Sources of individual problems are understood and eliminated.	Common sources of problems are understood and eliminated.
People	Success depends on individual heroics.	Success depends on individuals; management system supports.	Project groups work together, perhaps as an integrated product team.	Strong sense of teamwork exists within each project.	Strong sense of teamwork exists across the organization
	"Firefighting" is a way of life.	Commitments are understood and managed.	Training is planned and provided according to roles.		Everyone is involved in process improvement.
	Relationships between disciplines are uncoordinated, perhaps even adversarial.	People are trained.			
Technology	Introduction of new technology is risky.	Technology supports established, stable activities.	New technologies are evaluated on a qualitative basis.	New technologies are evaluated on a quantitative basis.	New technologies are proactively pursued and deployed.
Measurement	Data collection and analysis is ad hoc.	Planning and management data used by individual projects.	Data are collected and used in all defined processes.	Data definition and collection are standardized across the organization.	Data are used to evaluate and select process improvements.
			Data are systematically shared across projects.	Data are used to understand the process quantitatively and stabilize it.	

95-357 drw 12B

© Carnegie Mellon, Software Engineering Institute, Software Development Capability Maturity Model (CMM)

3.2.1.1 Why is this important to Systems Engineering?

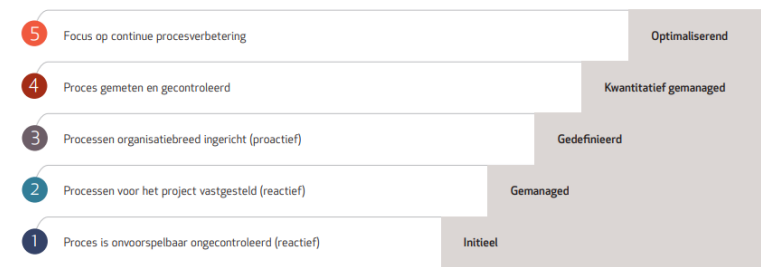
First of all, don't (just) take my word for it. The CMM(I) is also described in the "[Leidraad Systems Engineering](#)"²[5], meant for civil engineering projects:

² https://www.leidraadse.nl/assets/files/leidraaddownload/Leidraad_V3_SE_web.pdf

1.3 WERKEN MET EEN STAPPENPLAN

De omvang van GWW-projecten neemt toe in een omgeving waar met minder tijd en geld, veiliger en duurzamer moet worden gebouwd. Daarbij zijn vaak veel belanghebbenden betrokken bij projecten. Dit vraagt om prestatieverbetering van de betrokken organisaties. De afgelopen jaren zien we een ontwikkeling waarbij technologie, standaardisering en hulpmiddelen een belangrijke rol spelen in de zoektocht naar prestatieverbetering. Daarnaast kan deze verbetering worden gezocht in procesverbetering en optimalisatie van de aanwezige competenties bij de personeelssamenstelling. Deel II van deze Leidraad gaat in op het optimaliseren van de competenties (2.4). Een stappenplan kan een rol spelen bij de procesverbetering.

In de GWW-sector komen verschillende contracten op de markt. Daarbij bestaat er grote diversiteit in de omvang van contracten, complexiteit en bijvoorbeeld aantal betrokken belanghebbenden. Wanneer organisaties een strategie bepalen is het raadzaam dat ze vaststellen op welke contracten ze zich willen richten. Dit geldt zowel voor organisaties die contracten op de markt zetten als voor organisaties die daarop inschrijven. Wanneer de gekozen strategie om ontwikkeling vraagt, kan men hierbij een stappenplan (roadmap) gebruiken. Hierin legt de organisatie bijvoorbeeld vast welke markt ze wil bedienen en welke contracten daarbij passen, maar ook welk type medewerkers en welke kwaliteit van processen daarvoor nodig zijn. Vraagt de gekozen strategie om verbetering



Figuur 2 - Stappen in volwassenheidsniveau van een organisatie

van de processen, dan kan hierbij het stappenplan worden ingezet. Hiervoor gebruikt de sector vaak het CMMI-model (Capability Maturity Model Integration).

CMMI-model

Het door het Carnegie Mellon Software Engineering Institute ontwikkelde CMMI-model beschrijft een raamwerk van karakteristieke onderdelen van een effectief proces. De processen van een organisatie kunnen worden getoetst aan de hand van dit raamwerk, dat is afgeleid uit succesvolle praktijkervaringen. Daarbij wordt onderscheid gemaakt tussen een gefaseerde weergave en een continue weergave. Bij de gefaseerde weergave toetst men per niveau of de vastgestelde groep procesgebieden voor dat

niveau (Maturity-niveau) op orde is; in de continue weergave wordt per proces afzonderlijk een niveau toegekend (Capability-niveau). In huidige contracten wordt soms van afzonderlijke processen een bepaald Capability-niveau gevraagd. De ISO 15504 beschrijft een raamwerk waarmee deze afzonderlijke processen kunnen worden getoetst. De strekking van de niveaus van beide weergaves zijn vergelijkbaar; in figuur 2 zijn de 5 Maturity-niveaus weergegeven. De continue weergave hanteert ook nog een niveau 0 voor een incompleet proces.

Verhogen volwassenheidsniveau

De keuze voor het te bereiken volwassenheidsniveau moet passen binnen de strategie van de betreffende organisatie.

Second, think about what level your current Systems Engineering practice is, and what you would like it to be in your organization. Level 1 is probably not your goal, as everything is “panic-driven”. Level 5 seems very nice, but it’s very, very organized, maybe too “stiff” for a company that needs more agility and freedom to make quick choices (see Organizational Thinking).

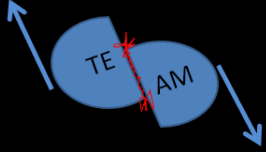
The third aspect that is important is to realize that different people in different companies act on different CMM levels. Being “mechanical engineer” for ASML is a completely different story than being a “mechanical engineer” for a small engineering bureau. The first will work in an organization of Level 4-5, and will have very tight ties with the company tools and regulations. The latter will have more liberty, more contact with the people they are working for, but also less structure in their job; maybe the organization is Level 1-2. The realization of the different levels is especially important when considering the deliverables of a project, particularly for things like certification and documentation. This could lead to a better realization of what is required to collaborate successfully.

To let these people talk about the deliverables of their “mechanical engineering” you’d need to find common ground, which is something the Systems Engineer could do, maybe by explicitly showing the difference in Capability Maturity. This helps to mutually estimate the amount of work, the skills and experience needed in the team, and in general expectation management.

Having different CMM between vendors (or colleagues) you can get (from[4]):

Capability Maturity Mismatch

When you and your vendor have different capability maturity levels... there can be a disruptive shearing effect on project processes



Capability Maturity Mismatch

Vendors say they see

- conflicting institutional voices/opinions (client doesn't speak with one voice)
- adversarial relationships ("I don't feel like we're on the same team")
- wrong people in key positions
- unrealistic expectations
- content-approval deadlines are not met
- undefined decision-making processes
- little or no measurement of key performance indicators
- insufficient staffing for the task at hand
- completed projects are not maintained after delivery

3.2.1.1.1 A small note on Level 1....

Sometimes, having no process can boost innovation, but often this is not sustainable. I can recommend reading "*Skunk Works: A Personal Memoir of My Years at Lockheed*"[6] about the cross-sectional team of "individual heroes" that was pulled out of Lockheed Martin to create the secret "Skunk Works" team to design the "impossible to fly" F-117 fighter plane. It's an action novel for engineers, and shows both opportunities and threats for this type of organization.

3.2.2 References

1. Paulk, Mark C., Bill Curtis, Mary B. Chrissis, and Charles V. Weber. "Capability Maturity Model for Software, Version 1.1." Fort Belvoir, VA: Defense Technical Information Center, February 1, 1993. <https://doi.org/10.21236/ADA263403>.
2. Michael Edson. "Good Projects Gone Bad: An Introduction to Process Maturity." 14:18:59 UTC. <https://www.slideshare.net/edsonm/good-projects-gone-bad-an-introduction-to-process-maturity>.
3. Paulk, Mark C., ed. *The Capability Maturity Model: Guidelines for Improving the Software Process*. The SEI Series in Software Engineering. Reading, Mass: Addison-Wesley Pub. Co, 1995.
4. "Good Projects Gone Bad: An Introduction to Process Maturity - [PPT Powerpoint]." Accessed November 30, 2021. <https://cupdf.com/document/good-projects-gone-bad-an-introduction-to-process-maturity-5584a0437a32e.html>.
5. "Welkom | Leidraad Voor Systems Engineering." Accessed December 1, 2021. <https://www.leidraadse.nl/>.
6. Rich, Ben R., and Leo Janos. *Skunk Works: A Personal Memoir of My Years at Lockheed*. 1. paperback ed. A Back Bay Book Military History, Technology. Boston: Little, Brown, 1994.

Chapter 4 Systems Design Tools

4.1 Addition to 4.7 and 4.13 ; Architecture Decision Records

The focus in 4.7 is on Architectures, the focus of 4.13 is on documentation.

Large parts of the Systems Engineering work is done on legacy systems; upgrades have to be made to existing systems, or even duplications need to be made while original parts are no longer available. It is important to be able to get clear what "forces" were into play when the original system was built, and what forces are at play now. For instance, a polluting but reliable diesel engine was no problem 10 years back, especially because electrical engines, drives and the battery systems were not as powerful then. When (re-) designing now, what new solution should be chosen?

Architecture Decision Records (ADRs) can be used to either get clarity on a current design, or to reverse architect an existing system. In "Systems Design and Engineering" not much is being said about dealing with legacy systems, or how to make sure that your systems design is going to be valuable to engineers working on your project in the future. This is why I'd like to discuss "Architecture Decision Records" and some related tools.

4.1.1 What is an ADR?

From Nygards blog[1]:

One of the hardest things to track during the life of a project is the motivation behind certain decisions. A new person coming on to a project may be perplexed, baffled, delighted, or infuriated by some past decision. Without understanding the rationale or consequences, this person has only two choices:

** Blindly accept the decision*

** Blindly change it*

It's better to avoid either blind acceptance or blind reversal.

An Architecture Decision Record is a document of 1-2 pages that describes an architectural decision. An architectural decision is a decision that impacts multiple system aspects (such as maintainability, performance, reliability). Examples are: "SQL database will be used for all data storage", "Robot will use differential drive" or "All parts should be manufacturable in our own workshop". The general format is:

Title

Status

What is the status, such as proposed, accepted, rejected, deprecated, superseded, etc.?

Context

What is the issue that we're seeing that is motivating this decision or change?

Decision

What is the change that we're proposing and/or doing?

Consequences

What becomes easier or more difficult to do because of this change?

The "rules" for ADRs coming from the original proposal by Nygard³ [1] are:

- ADRs are reviewed by the design team.
- After an ADR is accepted, it is immutable, no changes can be made.
- The decision is voiced in an active tense (not: "robots should be made serviceable" but "we will make robots serviceable").
- When a decision is changed, the previous ADR is marked as "superseded", and a new ADR is made with the new decision, and the new Context & Consequences

Over time, multiple formats have been created, among with [tools to maintain them](#)⁴[2]. There is a lot of background information on why it is good to make ADRs, among which Articles [from IEEE Focus](#)⁵[3] and a [very instructional video from IBM](#)⁶ [4].

In software "best practices" these ADRs are stored in the code repository. For hardware use, the decision of where to put them may need some team deliberation....

0007-De AGV krijgt 4 wielen waarvan 2 aangedreven met diff drive

VS Created by Victor Sluiter
Last updated: Jul 24, 2020 • 2 min read • 3 people viewed

Status:	Accepted
Last Changed Date:	28 Nov 2019

Context

De AGV wordt aangedreven met wielen, maar op welke manier? Er zijn verschillende opties te bedenken, ieder met voor- en nadelen. Een overzicht:

	Pro	Con
4 wheel	<ul style="list-style-type: none"> • Simple 	<ul style="list-style-type: none"> • Driving forward or backward requires different steering • Not included by default in navigation stacks?
6 wheel	<ul style="list-style-type: none"> • Driving forward or backward is the same 	<ul style="list-style-type: none"> • Might require spring loading of wheels to keep center wheel in contact with bumpy terrain.
4 wheel skid	<ul style="list-style-type: none"> • Driving forward or backward is the same 	<ul style="list-style-type: none"> • Cost (4 motor controllers, or complex mechanics) • Accuracy of skid steering in tight corners is highly dependent on floor friction

Een andere optie die door Wewo genoemd was, lijkt op een 'straddle carrier', waarbij twee wielen actief gestuurd worden en het derde stationair of een zwenkwiel is:



³ <https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions>

⁴ <https://github.com/joelparkerhenderson/architecture-decision-record>

⁵ https://personal.utdallas.edu/~chung/SA/zz-Impreso-architecture_decisions-tyree-05.pdf

⁶ https://youtu.be/41NVge3_cYo



Voordeel is dat je altijd contact van de wielen met de grond hebt (want 3 wielen). Nadeel is dat je de wielen actief moet sturen wat weer ruimte kost en duur is (4 drives i.p.v. 2).

Decision

We gaan een AGV bouwen met 4 wielen en differential drive (optie 1).

Consequences

Dit is het simpelste ontwerp, mechanisch. Dit betekent dat er meer aandacht aan het hoofdprobleem geschonken kan worden, de navigatie. Met de andere ontwerpen zou het maken van de ophanging / aansturing waarschijnlijk meer tijd kosten en een groot projectrisico vormen. De optie met skid steering is goedkoop en simpel, maar levert ook veel wrijving en daardoor onnauwkeurigheid op in het navigeren.

Nadeel aan 4-wiel is dat 1 aandrijfwiel in de lucht kan blijven hangen bij een hobbelige vloer. Een oplossing kan zijn om de constructie een beetje 'torsieslap' te maken.

Figure 5 Example Architecture Decision Record

4.1.2 Using Architecture decisions for reverse architecting and reviewing

When using ADRs is a known practice, this can be used to review current architectures for decisions that are no longer valid (updated insights from design team, or altered legislation, or changed viewpoint from customer, or). This review method can also be used to reverse architect an existing product, i.e. determine the decisions that were made when the product was made, and write these down. It is of course very helpful to have one of the original contributors to the product available when doing this.

A systematic way of doing this is the Decision Centric Architecture Review (DCAR) [method](http://www.dcar-evaluation.com)⁷[5], also described in [IEEE Focus](https://www.vanheesch.net/papers/dcar-ieeeSW.pdf)⁸[6].

4.1.3 Related subjects

4.1.3.1 Technical Debt

A subject that was discussed with partners quite often is that at the end of a development process a product was in the field, but maybe not fully verified, or not fully documented. Or that changes to a robot "in the field" would be made to quickly get it going, but never document the change, impeding support and further development. This is called "technical debt". In [7]:

Technical debt is a metaphor for describing a design or implementation construct that is expedient in the short term, but that sets up a technical context that can make a future change more costly or impossible. Causes related to planning and management are protagonists among those responsible for creating technical debt. For example, tight schedules, competitiveness, changes in business prioritization, and business dynamics are responsible for creating a turbulent environment that leads to technical debt.

or [8]:

⁷ <http://www.dcar-evaluation.com>

⁸ <https://www.vanheesch.net/papers/dcar-ieeeSW.pdf>

The technical debt (TD) metaphor, coined by Cunningham [1], has been used to describe the trade-off between short-term benefits gained by delaying certain development activities and the costs of implementing these activities in the future.

ADRs might help to quickly capture important decisions in the architecture, and thereby reduce the technical debt in a quick and acceptable way.

4.1.3.2 ARC42

If a larger framework is needed to store the Architectural Decisions in, ARC42 [9] can be used. ARC42 is an open standard to document (software) architecture. In this framework you can find the Architecture Decisions in section 9.

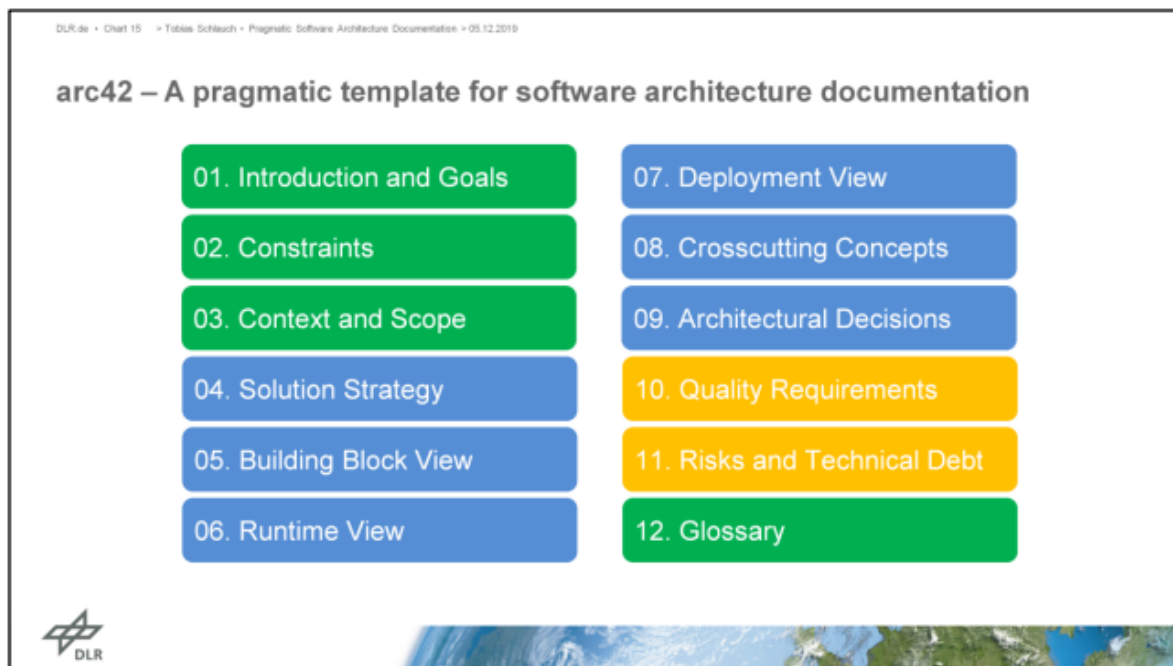


Figure 6 Overview of ARC42 documentation template

4.1.4 References

1. Nygard, Michael. "Documenting Architecture Decisions." *Cognitect.com*⁹, 00:00 500. <https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions>.
2. Henderson, Joel Parker. *Architecture Decision Record (ADR)*, 2021. <https://github.com/joelparkerhenderson/architecture-decision-record>.
3. Tyree, J., and A. Akerman. "Architecture Decisions: Demystifying Architecture." *IEEE Software* 22, no. 2 (March 2005): 19–27. <https://doi.org/10.1109/MS.2005.27>.
4. Software Engineering Institute, Carnegie Mellon University. *SATURN 2017 Talk: Architecture Decision Records in Action*, 2017. https://www.youtube.com/watch?v=41NVge3_cYo.
5. uwe. "DCAR – Decision-Centric Architecture Review | A Lightweight Method for Software Architecture Evaluation." Accessed November 10, 2021. <http://www.dcar-evaluation.com/>.
6. Heesch, Uwe van, Veli-Pekka Eloranta, Paris Avgeriou, Kai Koskimies, and Neil Harrison. "Decision-Centric Architecture Reviews." *IEEE Software* 31, no. 1 (January 2014): 69–76. <https://doi.org/10.1109/MS.2013.22>.
7. Rebouças de Almeida, Rodrigo, Rafael Ribeiro, Christoph Treude, and Uirá Kulesza. *Business-Driven Technical Debt Prioritization: An Industrial Case Study*, 2020.
8. Rebouças de Almeida, Rodrigo, Uirá Kulesza, Christoph Treude, D'angellys Feitosa, and Aliandro Lima. *Aligning Technical Debt Prioritization with Business Objectives: A Multiple-Case Study*, 2018. <https://doi.org/10.13140/RG.2.2.16106.21441>

⁹ <http://Cognitect.com>

9. Starke, Dr Gernot. "Arc42 Template Overview." arc42. Accessed November 10, 2021. <https://arc42.org/overview/> .

4.2 Addition to 4.9, FunKey Architecting and ValueFirst

The book describes how needs from stakeholders are explicitly addressed by FunKey. Although the examples are really clear in the book "Systems Design and Engineering", this aspect is not directly highlighted in the original thesis[1] or paper[2]. The description as in the book did remind me of the methodology described in "Competitive Engineering[3]" and other resources that describe both the requirements language "Planguage" and the "ValueFirst" method that deals with how Functions, Qualities and stakeholders interact.

Before diving into technicalities I'd like to share an article from Bits& Chips Magazine: <https://bits-chips.nl/artikel/system-requirements-defined-by-cascades-of-creativity/> [4] . It explains the "way of thinking" behind Competitive Engineering quite well. Cees Michielsens worked at ASML and DAF, and although he doesn't mention Gilb in this interview, he has acknowledged in personal conversation that his insights are strongly based on the Gilb methodology.

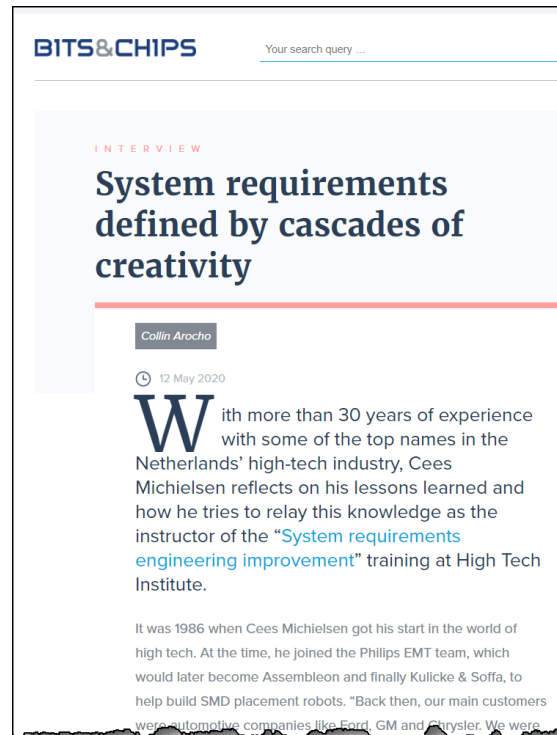


Figure 7 Article from Bits & Chips Magazine

4.2.1 Explanation of ValueFirst / Planguage

ValueFirst is driven by quantified requirements. For a more thorough explanation please see the section on [Planguage](#)(see page 30) .

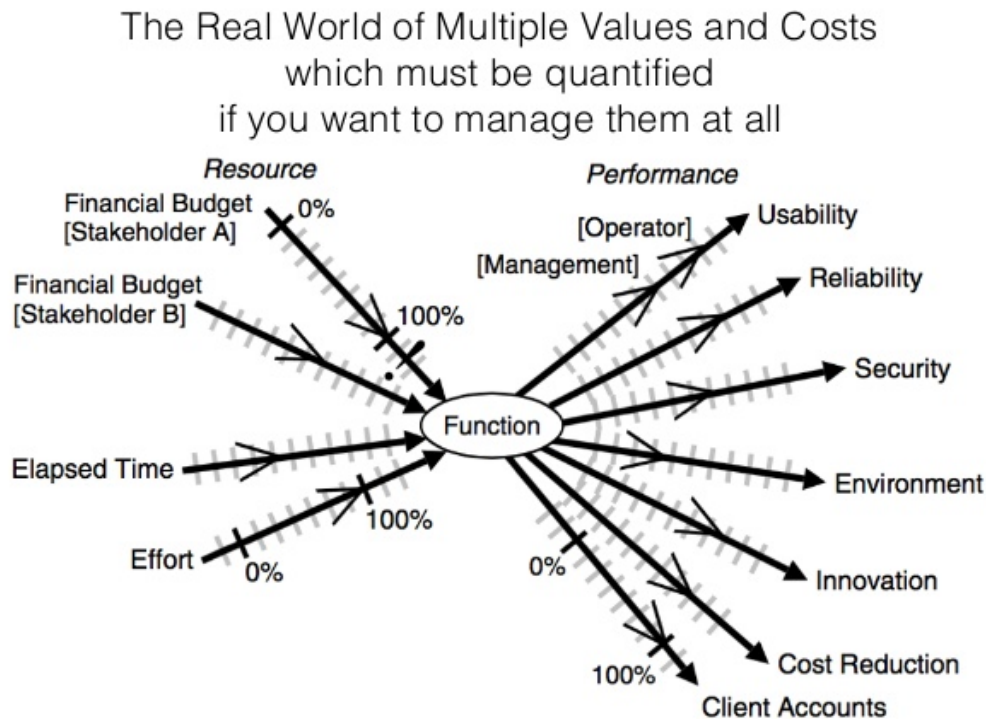


Figure 8 How performance qualities relate to functions. Source: [3]

4.2.1.1 Another view on Functions...

The view of "Functions" in Competitive Engineering / ValueFirst seems different than the view of Functions in FunKey. In ValueFirst, the definition of a Function is "What the System Does". But that is usually taken quite shallow, a car will mainly have a function "transporting from A to B". The focus is much more on the performance criteria (values) that are seen as criteria "how good a system is at doing its function". In this example, values could be how maintenance friendly the car is, or what the expected cost per km is, the maneuverability of the car, or how good it impresses the neighbors. All these "values" are the items that you *actually* take into mind when buying a car.

4.2.1.2 ... And how values "trickle down" to solutions ...

When "transporting from A to B" is the function of a car, multiple solutions can be found to do that, for instance using an electric car, or a second hand diesel car, or a new petrol car. All these solutions contribute in some way to the values of the car as described above, but also have their own values. For instance, maneuverability could be split into different subtasks, or in different values such as "ease of steering" and "visibility of environment". This last value could be increased by adding a new sub-solution which is a rearview camera. This way, it is possible to keep track of why something is important for the overall design of the product. It also makes it possible to quickly explain to a designer of a sub-sub-part of a system why it "has to look great" or "be very cheap" (probably both 😞). The importance of this way of thinking is also highlighted by Cees Michielsen in [4]:

During his training session, Michielsen explains that, in a system, the highest layer of abstraction is the level with the most general requirements, ie the system needs to be fast or have a certain look. But as you go down deeper into the system, it gets much more detailed. Suddenly, the layers are referring to different subjects or using different languages to express the requirements, which can be a little tricky for engineers to keep the information flowing.

"That's the real objective of requirements engineering, finding different ways to ensure that the data continues to cascade from top to bottom and from stakeholder needs to implementation, all without

losing any information,” suggests Michielsens. “I think if I were to summarize the challenge for requirements engineering, I would say that it lies mainly in the cascading of information throughout each abstraction or decomposition layer.”

In ValPlan, a commercial tool to track these value requirements, stakeholders and solutions, connection looks like this for a project where a product is made for robotizing a part of the firefighting tasks:

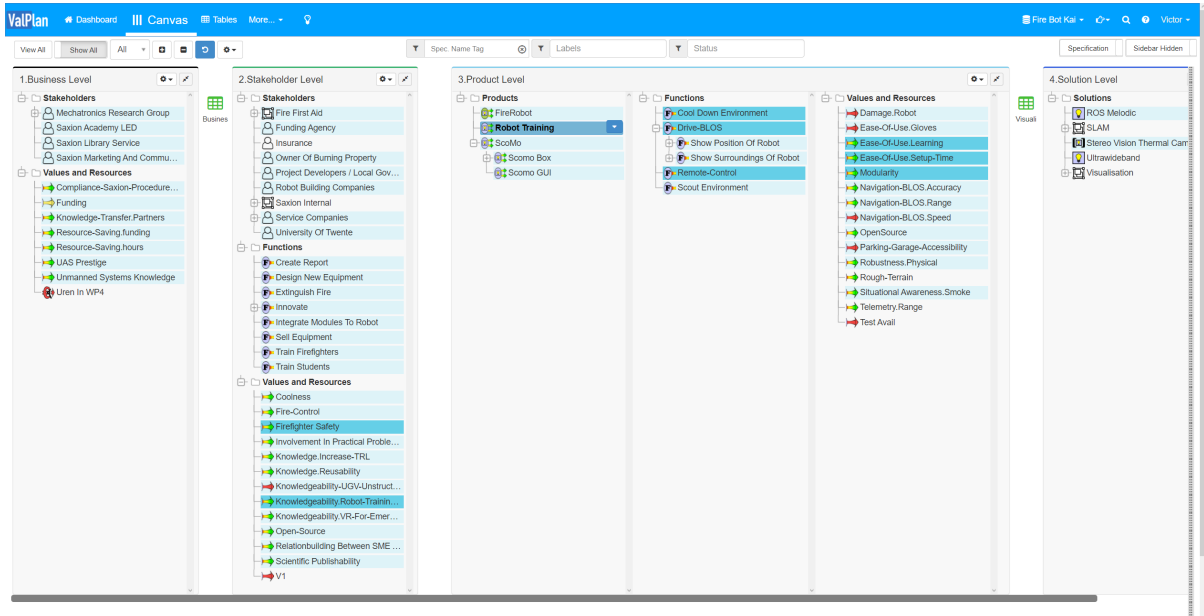


Figure 9 Screenshot of ValPlan

Looking at the stakeholder level, the “functions” of the stakeholder are “what the stakeholder does”. In the case of Saxion it’s “Innovate” and “Create Reports”, in the case of the Firefighters it’s “Extinguish Fire”, “Train Firefighters” and “innovate”. Values on this level are “Fire-Control” and “Open Source”. The connection of a Value to a Function is a “boolean”, this could be seen as the \times in the FunKey table.

The interesting part is that we can also see how at the Product Level solutions contribute to values on the Stakeholder level. One of the lessons we learned while making this overview is that we could make a very nice technical product, but that we do have to provide training for firefighters in order to make it usable in the field.

At the product level, the products again have functions and those functions have values, being realized by solutions at the “Solutions” level. Of course the model can extend further both to the left and to the right. What is important to realize is that a solution at level $n+1$ for a value in level n is a fixed functionality at level $n+2$. For most people in the project, knowing the Products, Functions, Values of their own level, the level below them and the level above them is enough. Systems Engineers might want to take one level more...



4.2.1.3 ... Leads to comparison matrices!

Because solutions contribute to many values we have to make tradeoffs. The Value Estimation Table can give insight in the efficiency of a solution. Because of the quantification of the values, we can see how completely different solutions compare or contribute to those values. Not all solutions and all values need to be in this table, multiple tables can be made for different subselections. Although the values are quantified, at this stage the numbers do not need to have scientific accuracy, as long as the source of the quantification is clear. The design team can use the table as a discussion piece.

This technique was used in the Nena project to choose what prototypes needed to be made to quickly test some requirements. We could have used an older robot and rebuilt that, which would have helped in testing navigation abilities, but would not have helped in judging whether the overall design would

be sized right, and no tests with the intended drive train could be done. The output of the table was that a prototyped aluminum bar frame would give the most "value" for little cost.

Strategy Comparison: Apples and Oranges

	Apples	Oranges	Alternative Strategies
<i>Objectives</i>			
Eater Acceptance From 50% to 80% of People	70%	85%	
Pesticide Measurement Reduce from 5% to 1%	50%	100%	
Shelf-Life Increase from 1 week to 1 month	70%	200%	
Vitamin C Increase from 50 mg to 100 mg per day	50%	80%	
Carbohydrates Increase from 100 mg to 200 mg per day	20%	5%	
Sum of Performance	260%	470%	
<i>Resources</i>			
Relative Cost Local currency	0.50	3.00	
Sum of Costs	0.50	3.00	
Performance to Cost Ratio	5.2	1.57	

"Evidence" for these numbers should, of course, be available on a separate sheet (but not shown here)

Figure 9.3

Comparison of Apples and Oranges using an IE table. IE allows you to compare all kinds of strategies (solutions) against your requirements.

Figure 10 "Impact Estimation Table" recently re-dubbed as "Value Estimation Table". This table uses the quantified values (here "objectives") and costs to make a Performance to Cost Estimation. Source: [3]

Note that these tables are different from FunKey coupling matrices. These are not used as much for architecting, where functions are coupled to key requirements (see section above), but to couple key requirements (values) to potential (partial) solutions.

4.2.2 References

1. Bonnema, Gerrit Maarten. "FunKey Architecting: An Integrated Approach to System Architecting Using Functions, Key Drivers and System Budgets," April 3, 2008. <https://research.utwente.nl/en/publications/funkey-architecting-an-integrated-approach-to-system-architecting>
2. Bonnema, G. Maarten. "Insight, Innovation, and the Big Picture in System Design." *Systems Engineering* 14, no. 3 (2011): 223–38. <https://doi.org/10.1002/sys.20174>.
3. Gilb, Tom, and Lindsey Brodie. *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Oxford Burlington, MA: Butterworth-Heinemann, 2005.
4. "System Requirements Defined by Cascades of Creativity – Bits&Chips." Accessed November 29, 2021. <https://bits-chips.nl/artikel/system-requirements-defined-by-cascades-of-creativity/>.

4.3 Addition to 4.14, Simulation

Section 4.14 in "Systems Design and Engineering" is a bit "sparse" on simulation. It only deals with dynamic simulation and discusses numerical integration methods which are applicable to linear(ized) models.

When talking about Systems Engineering and simulation, other aspects come to mind too:

4.3.1 Other forms of simulation

- FEM analysis; a part is modeled by building it up out of many small elements that have a certain physical property. Depending on this property, the simulation can be used for many purposes. Examples are:
 - simulating the deformation of parts under load. Generally this is not done in dynamic fashion, but mechanical parts can be tested for edge cases of mechanical load distribution. The output is whether a part or part assembly is strong enough for what it is meant to do. Based on the output, parts can be made lighter, stronger, or another material can be chosen.
 - simulating thermal distribution in a product. For example, simulate whether a printed circuit board with power electronics can be used in a housing without natural convection, in a hot environment. How will the heat distribution be towards temperature sensitive analog circuits in the same housing?
- Computer program simulation; the software and hardware environment in which a piece of software is being used can be simulated using pre-programmed models. The behaviour of the computer code that is under design can be inspected with simulated stimuli / responses instead of the "real" environment. One of the most used terms for simulating the external environment is "hardware-in-the-loop simulation"
- A *mockup* is a simulation of part of the complete system, to elicit stakeholder feedback. Some products are almost entirely completely computer simulated for end users before they are realized in its physical form. An example is a traffic control system that was tested on operators with simulated camera views on the traffic systems (trains, cars, cyclists, traffic lights and a bridge) before the actual bridge was built. This way, the systems engineer can learn a lot about the requirements of the real product before large investments are made.
- In Design Exploration, one of the first phases of new development, "quick and dirty" models are being made to attain basic understanding of the problem at hand. Some system behaviour can sometimes also be described in small code snippets. One of the tools that can combine these small "knowledge snippets" and calculate back and forth between them using MonteCarlo parameter variation is [Rees DSE](https://www.reden.nl/software/rees-dse)¹⁰.

4.3.2 Digital Twin

Recently, the term "Digital Twin" has gotten a lot of attention. Although it is hard to give an accurate description, and the term is used for different kinds of simulation, the general gist is that a "real" product has a simulated "twin" in simulation. Some of these twins are purely simulation, others are able to use inputs from the physical world and simulate the intended behaviour. The difference between an "standalone" simulation of one specific part of the product is that the "digital twin" is able to simulate across technical domains. Using [Model Based Systems Engineering](#)(see page 26) will help in creating and maintaining a digital twin. A [recent article in Mechatronica & Machinebouw](#)¹¹ [1] highlights that making a digital twin also needs a lot of investment of the organization to let multiple disciplines communicate with each other; a job well suited to a systems engineer.

¹⁰ <https://www.reden.nl/software/rees-dse>

¹¹ <https://mechatronicamachinebouw.nl/artikel/perspective-en-qing-zetten-schouders-onder-versnelling-digital-twinning/>.

4.3.3 References

1. "Perspective en Qing zetten schouders onder versnelling digital twinning – Mechatronica&Machinebouw." Accessed November 12, 2021. <https://mechatronicamachinebouw.nl/artikel/perspective-en-qing-zetten-schouders-onder-versnelling-digital-twinning/>.
- 2.

4.4 Model Based Systems Engineering

The subject of Model Based Systems Engineering (MBSE) is not discussed in the book Systems Design and Engineering, but it has taken such a flight in the recent years that I felt compelled to discuss it here.

4.4.1 MBSE in short

The subject of MBSE is a very large one, and it's hard to thoroughly in a few words. Despite that, I'll try.

In traditional Systems Engineering, documents are the captivation of knowledge about the system. Requirements documents, Functional Design Documents, Interface Descriptions and Test reports all describe how a system was designed, and what the relations of those systems are. The more complex a product becomes, the harder it is to keep all those documents "in sync" and the harder it is to find the information that you need, or to get a good overview of how a certain parts are related to each other. In Model Based Systems Engineering, the "model" of the system is the "single source of truth" of everything that relates to the system under design. Documents can be made, but these are artefacts, generated **from** the model.

Just to be clear, in general a "model" here is a "boxes and lines" model, not a 3D model or a dynamic model, although those should ideally be driven by the properties stored in the MBSE model. The MBSE model's purpose is to contain all information about connections, sequences and other relations between all parts in the system, be it electrical, mechanical, hydraulics or software. In many cases, engineers already have this knowledge, and mostly already have some diagrams describing the relations. However, these diagrams usually are not consistent between each other. The purpose of MBSE is to have diagrams that are **always** consistent, and that for different purposes (stakeholder analysis, insight in an engineering problem) different diagrams can be made that highlight different system aspects.

To put it in another way, see this introduction from the paper "The Long and Winding Road: MBSE Adoption for Functional Avionics of Spacecraft." [1]:

Interest in Model-Based Systems Engineering (MBSE) over the traditional approach to systems engineering, Document-Based Systems Engineering (DBSE), is growing. With DBSE, project and design information is stored in documents and must be manually maintained and transferred between domains. The traditional DBSE approach¹² is labour-intensive and consists mostly of manual analysis, review and inspection.

MBSE is the formalised application of modelling to support system requirements, design, analysis, optimisation, verification and validation. By using interconnected models to store, represent and relate this information and data, projects can expect improvements in consistency, communication, clarity, visibility, maintainability¹³, etc. – thus addressing issues associated with cost, complexity and safety.

To create these models, a modelling language is needed. To create the diagrams and check for internal consistency, a tool is needed that can check the consistency of the diagrams, and export

¹² <https://www-sciencedirect-com.saxion.idm.oclc.org/topics/computer-science/system-engineering-approach>

¹³ <https://www-sciencedirect-com.saxion.idm.oclc.org/topics/computer-science/maintainability>

information to third party tools. To know what kind of diagram is needed for what kind of information, a method is needed. From [MBSE for Dummies](#)¹⁴[2]:

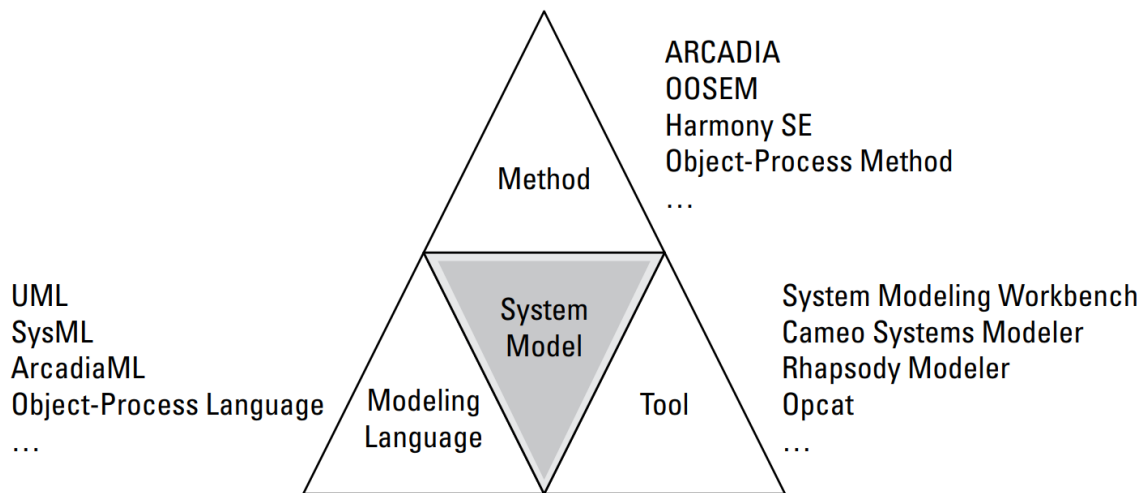


FIGURE 2-2: MBSE Essentials Pyramid.

Figure 11 Examples of Tools, Modelling Languages and Methods for MBSE. Source: [2]

Different languages need different tools, and have different focus points. For example, the Arcadia method, supported by the open source Capella tool is focused on keeping an “explainable” architecture view. The lowest level it deals with is “physical level”, which are individual components. But to add “real physical parameters” such as for example wheel friction or steering radii of robots and how those affect driving behavior, a plugin is needed and this is outside the Arcadia “method”. On the other hand, the SysML language is very good at making low level physics models, but does not have a single clear method to model the system. Because it originates from UML, it also is highly based on concepts from object oriented software engineering such as inheritance and encapsulation, concepts that are nonexistent in Arcadia / Capella.

4.4.2 Benefits and caveats

The benefit of using MBSE is that it becomes easier to communicate system behavior. The systems engineer can use the model to verify that engineers understand the relations between their disciplines, and to communicate (expected) system behavior with stakeholders. Also, because the engineers “feed back” information about how systems are realized, it is possible to let the model check whether certain scenarios remain feasible.

¹⁴https://static.sw.cdn.siemens.com/siemens-disw-assets/public/ny8JDhe7Nsrg3dBdPRxL/en-US/MBSE---MBSE-For-Dummies_tcm27-101485.pdf



FIGURE 3-1: Managing parameters across multiple domains.

Figure 12 When fully embedded in the organization, this integration is possible using MBSE. Source: [2]

As already indicated both by Jon Holt [3] and Jenkins[4] a full implementation is not necessary. A company can start by modelling part of a product, or a certain abstraction level of the product.

A strong caveat of MBSE is therefore the effort needed to create and especially to maintain the model. Changes in any domain that is integrated into the model should be fed back to the model. Ideally, the CAD tools used for electrical and mechanical engineering should be directly coupled to the model, and the designs should only be “views” of the model in that specific domain. Some vendors try to do this as much as possible, for example Dassault has an integration between SysML and SolidWorks, and Siemens provides a full toolchain for their products. The drawback is that this is a very strong vendor lock-in, and the maintenance of these models will take time. Even if this is understood by the engineers who would like to use this tooling, the organization needs to change to use a “model centric” way of work. This is one of the reasons that MBSE is mostly used in very large organizations that deal with aerospace or medical (Lockheed Martin, Thales, ESA, NASA, Philips are strong promoters). These companies have many, many subcontractors but the need for safety and system maintenance is large enough to pay for the overhead needed to maintain a model.

Some open source and free options exist for MBSE, but most tools are commercial.

4.4.2.1 MBSE for SME

To my delight, ESA already investigated the usage of MBSE for small and medium enterprises [4]. Because ESA uses MBSE in their own missions, it would help them if subcontractors could also use MBSE. The full talk can be found here on Youtube: <https://youtu.be/5WzdVxMWm1U>.

Jenkins [4] sees [three scenarios for SMEs](#)¹⁵ and whether they should adopt MBSE or not:

¹⁵<https://indico.esa.int/event/386/contributions/6225/attachments/4267/6446/1105%20-%20MBSE%20in%20an%20SME%20Context.pdf>

Sce- nario	Company	Advise
1	Small company producing simple systems	<ul style="list-style-type: none"> • MBSE is NOT recommended • MBSE is not something that will add much value to the company • In a small team its easier to keep track of everyone's work • Information management is not a big problem
2	A company transitioning from single to multi-missions, increasing workforce and increased employment of grads/people early in their career and CAN afford the time and cost of adoption	<ul style="list-style-type: none"> • MBSE IS recommended • MBSE helps to enforce a standardised SE process • This helps to ensure that all outputs are consistent • Makes it easier to teach newcomers the company SE process • MBSE helps aid the SE process to ensure the system is well defined • MBSE helps to manage information – ensures traceability
3	A company transitioning from single to multi-missions, increasing workforce and increased employment of grads/people early in their career and CANNOT afford the time and cost of adoption.	<ul style="list-style-type: none"> • MBSE is NOT recommended • MBSE provides benefits to the SE process of an SME • However if the SME cannot afford time and cost of adopting MBSE then adoption could be detrimental • For MBSE to be accessible to SMEs I recommend the following needs to happen: <ul style="list-style-type: none"> • Accessibility to MBSE resources – case studies, suggested tools, strengths & weaknesses of tools • MBSE to mature – more people becoming experts, best way to use the tool • Proof of ROI • Mitigate vendor locking worry

4.4.3 References

1. Gregory, Joe, Lucy Berthoud, Theo Tryfonas, Alain Rossignol, and Ludovic Faure. "The Long and Winding Road: MBSE Adoption for Functional Avionics of Spacecraft." *Journal of Systems and Software* 160 (February 1, 2020): 110453. <https://doi.org/10.1016/j.jss.2019.110453> .
2. Kaelble, Steve. "MBSE For Dummies®, Siemens Special Edition," 2022, 53.
3. Swiss Society of Systems Engineering - SSSE. *Jon Holt - Scarecrow Consultants*, 2019. <https://www.youtube.com/watch?v=VnBWyc-Srqq> .
4. Jenkins, Rhiannon Caitlin. "MBSE in an SME Context," n.d., 17.

Chapter 5 Additional subjects

The following subjects did not directly fit into the "Thinking Tracks" and "Systems Design Tools", but were stumbled upon during the interviews, and therefore given a place in this section. Any subject discussed here could deserve its own complete book, but we're just scratching the surface here to highlight what these subjects could do for companies in the SESAME project.

5.1 Requirements Languages

In the interviews with companies we found that many problems arise from vague or incomplete requirements, or from getting many requirements that actually do not describe the customer wish. These problems might be alleviated by using more formal requirements languages.

These languages force the users to shape natural language, which is open for multiple interpretations, into more concise requirements. Of course this requires training of all stakeholders to use this more formal notation. The methods presented here are -from experience in the research group- easily learnable and transferrable.

5.1.1 Planguage

5.1.1.1 Explanation of ValueFirst / Planguage

When you need to buy a car and specify its "features" (has >3 doors, has a steering wheel, has a car radio...), you could find a car that does its function (transporting people from A to B) for less than EUR 200. Why do most of us spend more money? Because we don't necessarily want the features, we want to have the function, but with reliability, comfort, good looks, low cost per mile, Why do people pay \$1400 for an Apple Hermes watch instead of \$400 for a standard Apple Watch, while both have the same features? It's because they buy "style" or "exclusiveness". Planguage is the requirements language made to capture those subjective **values** and ValueFirst is the method to look at those functions and values from different product "levels". Please take a look at [Addition to 4.9, FunKey Architecting and ValueFirst](#)(see page 21) for an example project where this has been used on different levels.

The benefits of specifying (high level) requirements this way is that it sometimes becomes clear what other products might be necessary *besides* your first intuitive guess. For instance, we found out in a research project at Saxion that "Training time" was a very important parameter, and in order to test that, we also needed to make a clear instruction for usage of the equipment we developed.

The user requirements are quantified using "ValueFirst" methodology as described by Tom and Kai Gilb. In ValueFirst, the requirements of a function (here: chip alignment) are on the "values". The reason a user likes a product, is because the functions of the product have values that are interesting. For example, if you consider buying a new car you could buy the cheapest second hand one (which fits the requirement "4 wheels and a steer"), but you don't because you pay more for things like "style" or "reliability" or "comfort". Those things are the **values**, and they can be larger or smaller. The specification of these value requirements is done using Planguage. Each value is measured on a **scale**, and the way it is measured is called the **meter**. The **ambition** of each value is the "management summary" of what we want to achieve with the value. Terms can be found in the Gilb [glossary](#)¹⁶. On the scale, we can express what we want to achieve within this project (the **goal** level), but also what is the current state of the art (**record**) or at which value the function as a whole is so bad that even if all other values surpass the goal level, the function is not good (**fail** level. For example comfortable car which looks good but with a rotten chassis).

¹⁶ <http://concepts.gilb.com/Glossary>

To make the scale and meter reusable in different situations, **qualifiers** are used. These are the words in [brackets]. For example: Scale: [area] domination in defined [time], where a goal could be to have [area=world] and [time=one year] to say we'd like to have world domination within a year.

Resources can be expressed in the same way in order to make tradeoffs of which values are contributed to against which cost.

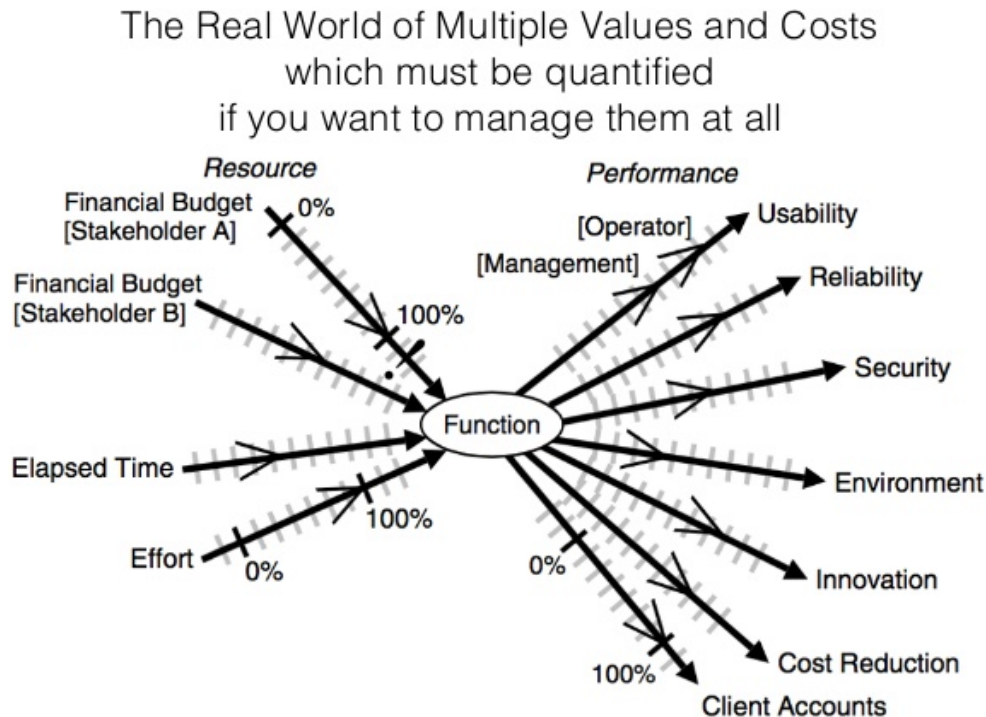


Figure 13 How performance qualities relate to functions. Source: [1]

5.1.1.2 Examples of research group

Here are some examples (slightly altered to hide project specific details) where the research group used Planguage to capture "User Requirements", the requirements that should be "validated".

5.1.1.2.1 TestCapability

Ambition: make sure that this product can do many, many types of tests, place one machine at the client, which can test most paint types.

Scale: scale of 0-10 whether [selected users] think that the chosen design of alignment and paint cartridge is suitable to expand to other tests, where
0=can only be used for a single paint type, device should be completely redesigned to do anything else than detect <known paint types>.

3 = to change test to other paint types, the alignment concept should be largely redesigned

5 = to change test to other paint types, the alignment concept is OK, but other parts (paint cartridge, test machine) need <major redesign>.

7 = will be able to replace current pigment paint based tests with less than 10 man year development time on paint cartridge.

10 = can comfortably test > 90% of paint types with less than 1 man year development time on cartridge once paint test type is known.

Meter: show the concept to [selected users] and ask to score based on the scale given. Average the results

[selected users] = 1 person of {Sigma, Gamma, Sikkens}

Goal: 5 ← Guess by Victor

5.1.1.2.2 Ease of Use - Operator Training

Stakeholders: Paint shops, Sikkens, Sigma

Ambition: Make sure that almost anyone can do a test with minimal training

Scale: Number of minutes training required to be able to do a <successful paint test> within a specified [time] using the [setup].

Meter: Record time from beginning of training activity until user is able to start a measurement with <successful> paint clarification in [setup] within [time] after receiving a sample. Average over 5 users.

Goal [setup=demonstrator, time=120s]: 10 min.

Past [setup = manual lab equipment, time=300s]: 2400 minutes ← Cindy, 40h training @ paint lab

5.1.1.3 Cheat sheet

ValueFirst Requirements CheatSheet

Version 22 Sep. 2018

Response.Time

Type: *Product Value*

Stakeholders: Users, Sales

Scale: milliseconds, from user enters a defined [Input-Command], until screen is updated correspondingly.

Past [Jan. 1 201x, Input-Command=Select-Contact] 600 ms <- Test a

Tolerable [Dec. 1 201x, Input-Command=Select-Contact] 200 ms <- Kai

Goal [Dec. 1 201x, Input-Command=Select-Contact] 60 ms ?? <- Kai

Stretch [Dec. 1 201x, Input-Command=Select-Contact] 45 ms ?? <- Kai

Name-Tag

Type: *Level + Requirement Type*

Stakeholders: List key interested people, groups or systems.

Scale: Quantification of value.

Past = defines any Past, like previous product.

Tolerable = defines point from failure to Tolerable

Goal = defines Success, Commitment or promise

Stretch = work towards but no commitment

[Conditions] number to Scale <- Source

Quantification Scale components.

Unit = any interesting unit. Examples: like time, €, number of, dB etc.

Rate = per something, so we can compare to see progress.

Examples, per customer, per day, per GB

Description = describe and put in context the use of the Unit and Rate.

Examples: to open a bank account, from being in front of a web-browser until account is created and user is logged into account.

Additional Parameters

Administration

Headline: = Summary text

Version: = can use **date and time** of edit

Owner: = only person or group **authorized to edit**.

Responsible: = responsible for making this requirement **happen**.

Meter: = how we intend to **measure/test** where we are on the Scale.

Wish = **Expressed** desired level that is **not** yet **agreed** upon.

Status = where we are at **last measurement**

Record = any interesting world Record

Optimum = perfect optimum level.

Trend [Marked] = where we think the competitors will be.

Trend [Internal] = what will happen in time to our system if we don't improve it

<- Specify the Source of that statement/number. First is you. Use ?? SWAG.

Levels

Business = Our Company

Stakeholder = People, Companies, Groups we serve.

Product = Highest level of our product or service

Sub-Product = A part of the Product

Solution = a "technical" solution.

Requirement Types

Value = **How Well** it does its Functions

Function = **What** the system/stakeholder does.

Solution = **How** it does it.

Development Resource = any resource used for development.



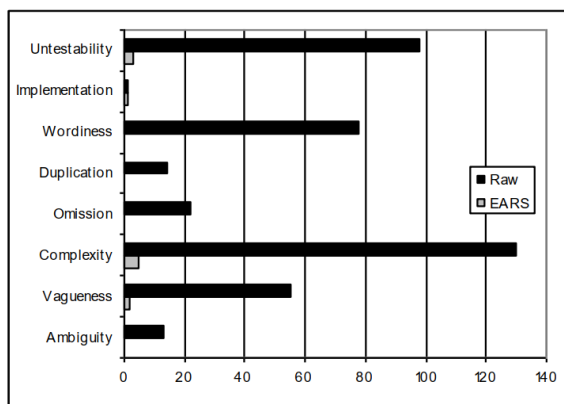
Figure 14 Used with permission by Kai Gilb, www.gilb.com

5.1.1.4 References

1. Gilb, Tom, and Lindsey Brodie. *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Oxford Burlington, MA: Butterworth-Heinemann, 2005.

5.1.2 EARS

The Easy Approach to Requirements Syntax was developed by Alistair Mavin from Rolls Royce airplane engine development. It was presented [to the public in 2009](#)¹⁷ [1]. The problem that the team from Rolls Royce was trying to solve was that requirements written down in Natural Language ("normal" English) suffer from many potential problems: ambiguity, vagueness, complexity, omission, duplication, wordiness, inappropriate implementation. Explanations for these terms are given in [1]. To solve this, Mavin made some simple rules on how to write requirements down, thereby slightly structuring the text. To the surprise of the team of Rolls Royce, this worked remarkably well, and the following year they published how rewriting a certification specification solved many of the original problems[2]:



Graph 1: Count of Problems in CS-E Requirements

<p>Raw Requirement: CS-E 590</p> <p>Where the starter is declared as part of the Engine, its design, and that of its associated drive mechanism, must be such that over-speeding of the starter, to an extent which could result in a Hazardous Engine Effect, cannot occur under any Fault conditions in the Engine which cannot be classified as Extremely Remote. The possibility of the starter remaining connected, or subsequently becoming reconnected, to the Engine resulting from any Failure of the drive system must be considered. Where in showing compliance with this paragraph, dependence is placed on safety provisions to be provided as part of the installation, the need for such provisions must be declared.</p>
<p>EARS Interpretation</p> <p>Where the control system includes the engine starter, the control system shall prevent starter overspeed which could result in a Hazardous Engine Effect.</p> <p>When considering engine starter drive failure mechanisms, the control system safety assessment shall assess the possibility of the starter remaining connected, or subsequently becoming reconnected, to the engine.</p>
<p>Notes</p> <p>The raw requirement is wordy and complex. The first interpreted requirement is an optional feature, since the ignition system is not always part of the control system. The second interpreted requirement is an event-driven requirement on the system safety assessment.</p>

Following the initial EARS publication, [many companies adopted it](#)¹⁸[3,5]. Success stories come from Toshiba Aerospace, IBM, Intel and even from partners in the nuclear domain. Some companies adopted EARS a bit[4], but most stayed with the basic rules:

¹⁷ https://www.researchgate.net/publication/224079416_Easy_approach_to_requirements_syntax_EARS

¹⁸ https://www.researchgate.net/publication/335535918_Ten_Years_of_EARS

INTRODUCTION TO THE EASY APPROACH TO REQUIREMENTS SYNTAX

The Easy Approach to Requirements Syntax (EARS) is a simple notation for writing textual requirements that was first published at the IEEE Requirements Engineering 2009 Conference.¹ The clauses of an EARS requirement are always in the same order and denoted by keywords. The generic EARS syntax is

- *While <optional precondition>, when <optional trigger>, the <system name> shall <system response>.*

Requirements written using EARS follow one of six simple patterns as described briefly here.

- 1) Ubiquitous. No EARS keyword. Continuously active.
 - *The mobile phone shall have a mass of fewer than XX grams.*
- 2) State driven. Keyword **While**. Active as long as some precondition remains true.
 - *While an external speaker is connected, the laptop shall mute the built-in speaker and send the audio output signal to the external speaker.*
- 3) Event driven. Keyword **When**. Activated by a discrete triggering event.
 - *When a card is inserted, the ATM shall verify the card.*
- 4) Optional feature. Keyword **Where**. Used to handle system or product variation.
 - *Where the car has a sunroof, the car shall have a sunroof control panel on the driver door.*
- 5) Unwanted behavior. Keywords **If** and **Then**. Required system response to unwanted events.
 - *If an invalid credit card number is entered, then the website shall display "please re-enter credit card details."*
- 6) Complex. Combinations of the previously mentioned keywords.
 - *While the aircraft is on ground, when reverse thrust is commanded, the engine control system shall enable reverse thrust.*

EARS is described in more detail in Mavin et al.¹ and Mavin and Wilkinson.³

Figure 15 EARS language rules. Source: [3]

The application of EARS in our own research group has led to a clearer way of writing requirements. If you want to implement EARS in your organization, it is worth reading the “[Listens learned](#)”¹⁹ paper[5] from Mavin.

5.1.2.1 Example from usage at the research group

In our own group, we use EARS, combined with a rationale that explains the reasons for the requirements (important!), and a MoSCoW rating (Must, Should, Could, Would). This way, a requirement can be written down that is “nice to have” (use sparingly).

¹⁹ https://www.researchgate.net/publication/308970788_Listens_Learned_8_Lessons_Learned_Applying_EARS

Requirements on function 5: User interaction

Number	Requirement	Rationale	Priority
FR0501	While in measurement phase, the demonstrator shall show the user the SNR of each output	Check requirements on SNR	M
FR0502	While in measurement phase, if loss-of-light is detected the demonstrator shall indicate the measurement as "failed"	See error requirements (FR063x) for readout for loss-of-light condition	M
FR0503	While a cartridge is in the demonstrator, the demonstrator shall enable the user to restart active alignment.		M
FR0504	When the cartridge is pushed into the measurement location, the operator shall be provided with clear tactile feedback that the cartridge is inserted correctly	"click" when cartridge is in correct place.	S
FR0505	When the cartridge is inserted, the demonstrator will start the alignment automatically within 1s.	"Ease of use" → start measurement procedure by aligning when inserting a cartridge	M
FR0506	When the alignment phase ends and no succesful alignment is detected, the demonstrator shall indicate an error.	No use to continue to measurement phase, show user that there is an error.	M

5.1.2.2 Cheatsheet from Aalto university

source: https://aaltodoc.aalto.fi/bitstream/handle/123456789/12861/D5_uusitalo_eero_2012.pdf

**EARS: Using combined sentences****Example: Optional feature combined with state-driven and event-driven**

- Where the car has an ABS system, while the car is moving, when the driver applies brake, the ABS system shall detect blocked wheels.
- When the ABS system detects a blocked wheel, the ABS system shall reduce effective brake pressure for that wheel until the wheel is unblocked.

Troubleshooting EARS problems**No sentence type fits!**

- Are you translating a requirement?

I can't identify the actor!

- Use a higher abstraction level until it makes sense
- Or get more information from relevant stakeholder

There's no system response!

- Usually the case with nonfunctional requirements
- Can be expressed as "the system shall be ..."

There's no template for "shall not"!

- Feature of EARS, try stating as "shall be immune" or similar workaround
- As last resort just use "shall not" structure

EARS produces too many atomic requirements!

- Deep technical requirements aren't well suited to EARS
- If necessary, use a list as accompaniment
- Consider other format for technical requirements if EARS seems inappropriate

Beyond EARS: Other good practices**Use a template that:**

- Provides for necessary metadata, e.g. requirement identifier
- Has provision for non-requirements, e.g. notes and examples
- But don't be dragged down by too heavy templates

Remember to keep your requirements up to date**Remember characteristics of good requirements****Requirements are about communicating between stakeholders**

- Ensure you can see the forest from the trees
- Methods aren't the meaning, they are a means to an end

SAFIR2014/SAREMAN project. www.cse.aalto.fi/SAREMAN

5.1.2.3 References

1. Mavin, Alistair, Philip Wilkinson, Adrian Harwood, and Mark Novak. *Easy Approach to Requirements Syntax (EARS)*, 2009. <https://doi.org/10.1109/RE.2009.9> .
2. Mavin, Alistair, and Philip Wilkinson. *Big Ears (The Return of Easy Approach to Requirements Engineering)*, 2010. <https://doi.org/10.1109/RE.2010.39> .
3. Mavin, Alistair, and Philip Wilkinson. "Ten Years of EARS." *IEEE Software* 36, no. 5 (September 2019): 10–14. <https://doi.org/10.1109/MS.2019.2921164> .
4. Bhatt, Devesh, Anitha Murugesan, Brendan Hall, Hao Ren, and Yogananda Jeppu. *The CLEAR Way To Transparent Formal Methods*, 2018. <https://doi.org/10.13140/RG.2.2.10946.89289>.
5. Mavin, Alistair, Philip Wilkinson, Sarah Gregory, and Eero Uusitalo. *Listens Learned (8 Lessons Learned Applying EARS)*, 2016. <https://doi.org/10.1109/RE.2016.38> .

5.2 Incremental Delivery of hardware

As discussed in [Addition to 2.3, A practical implementation of SE\(see page 8\)](#) , Systems Design and Systems Engineering describes the classic V-model approach. Many companies see the successes of agile software development (early releases, early customer feedback, quick turnaround on changing requirements), and wonder whether this can not be translated to hardware.

SCRUM as a method for agility (remember: [implementing SCRUM doesn't make you Agile](#)²⁰[1]!) does not transfer easily to hardware. It also suffers from being able to churn out lots of code with high velocity, but maybe not generating the right thing. This is a larger problem for hardware, where lead times and material costs are adding constraints on how often you can iterate within a time frame, something that is neglected by most software development methods.

After looking for methods to incrementally deliver and / or design hardware, I think two methods are fit for usage in small and medium enterprises because they

²⁰ <http://www.agilecio.net/blog/scrumb-is-eeen-methode-agile-eeen-mindset>

- are described in enough detail
- and have been proven to use in real hardware projects

These methods are EVO and RLC, as described in the next chapters.

Something worth noting, is that both these methods put considerable effort on the project leader to maintain progress and maintain structure. These methods cannot be used without commitment and knowledgeability of both the team members and their project leader.

5.2.1 References

1. Agile CIO. "Scrum Is Een Methode, Agile Is Een Mindset." Accessed December 22, 2021. <http://www.agilecio.net/blog/scrum-is-een-methode-agile-een-mindset>.

5.2.2 EVO

Although EVO is not very well known it is *predecesing* the SCRUM and the Agile Manifesto, it is "Agile avant la lettre". EVO is a project management **and** development methodology devised by Tom and Kai Gilb, and is described in Competitive Engineering[1]. As such, it heavily relies on Planguage, so please the [Planguage](#)(see page 30) and [Addition to 4.9, FunKey Architecting and ValueFirst](#)(see page 21) chapter for more background information. More explanation is given in booklets on the website of Niels Malotaux [2]. EVO is also described well in Larman's "Agile and Iterative Development: A Manager's Guide." [3], the chapter on EVO can be downloaded [here](#)²¹. To quote Larman:

[EVO] emphasizes—short iteration by iteration—making maximum progress towards the client's current highest-priority requirements, for the lowest cost. And each iteration, delivering into the hands of some stakeholders some useful results, so that early benefit and feedback is achieved. This is the practice of client-driven adaptive planning and evolutionary delivery.

The idea behind EVO is that you know what "values" your customer has (through Planguage requirements), and that you know how long your project will last. What EVO is trying to prevent is that a project is at its deadline, and nothing is delivered. The core "promise" is that value is generated every cycle. So your customer should get something "interesting" every 2-3 weeks, based on which you can check whether your mutual understanding of the values is still correct (if not, the customer is not happy with your delivery). When the project deadline has passed, and due to whatever circumstances the full product is not finished yet, the customer will already have some delivered (sub) systems that provide value.

5.2.2.1 How to "do" EVO?

There are several ways to look at EVO. For one, it's a Plan-Do-Check-Act cycle, on several levels. Each short EVO cycle of 1-2 weeks you check whether you can plan and deliver what you promised to deliver. Key here is that each cycle the team decides what to do to contribute to at least one of the values and *finishes that*. If you can't finish what you've promised, you have to promise less in the next cycle to adjust your estimate of how much you can do. These very short cycles are in fact short waterfalls.

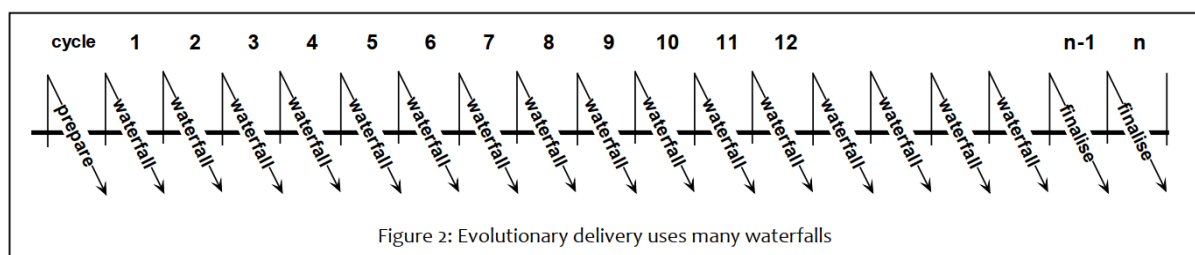


Figure 2: Evolutionary delivery uses many waterfalls

Figure 16 EVO cycles are small waterfalls. Source: [4]

²¹ <http://concepts.gilb.com/dl66>

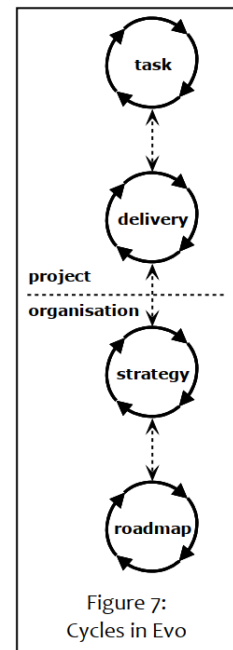


Figure 17 Cycles in Evo. Source:[4]

On a larger scale, by delivering value and checking the stakeholders' reactions, you can see whether your deliveries are creating value, see the cycles on the right.

Delivering hardware incrementally means that you have to plan ahead, and “deliverables” can also be simulation models, a mockup, or an addition to an existing product to test whether a new solution actually improves one of the values. In fact this is what Gilb actively promotes: try to get something physical as soon as possible to check whether your stakeholder’s values improve!

In order to continually create new delivery, some work has to be planned to happen in the “backroom”, invisible to the stakeholders. See figure below:

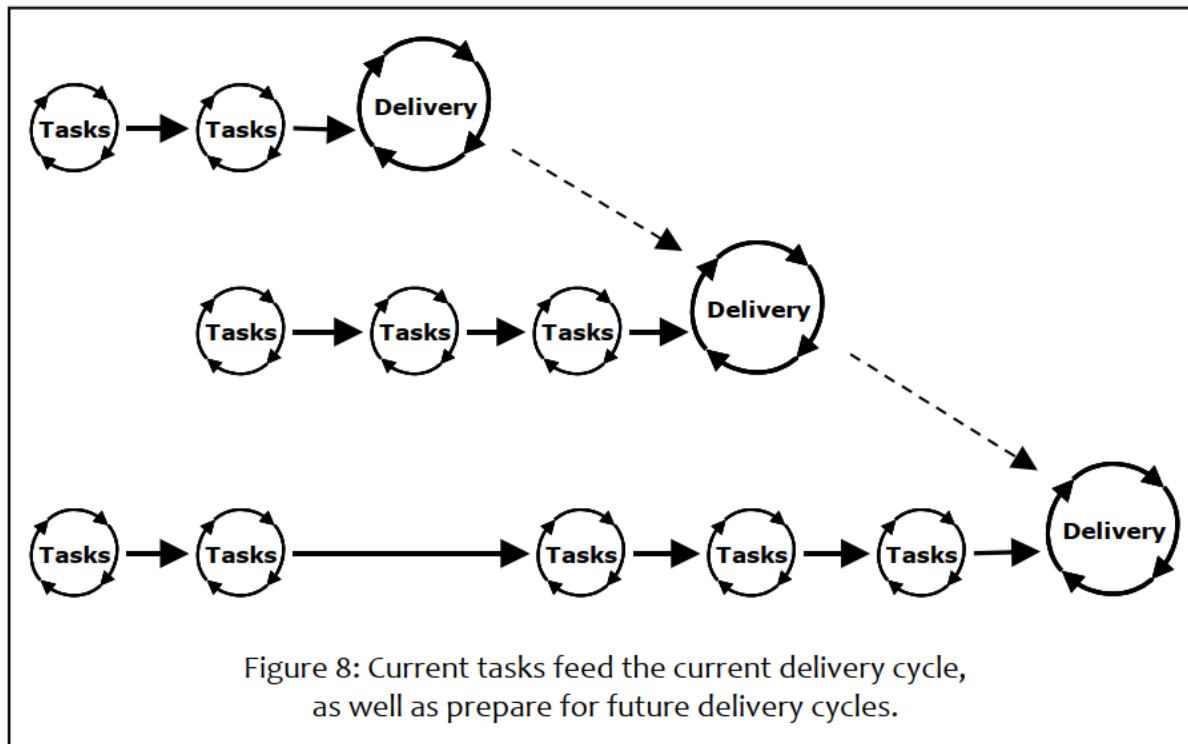


Figure 18 Some delivery needs to be prepared in earlier cycles. This is also planned in EVO.

One aspect of this is to know what solutions are create lots of value for a low effort. In the NENA project, an aluminum mockup was a quick and easy way to test some key assumptions. Other possibilities could have been to make changes to our existing robots, or to immediately create a simulation model. Choosing which solution to choose first is done by using Value Estimation Tables. See [Addition to 4.9, FunKey Architecting and ValueFirst\(see page 21\)](#) for an “apples against oranges” example, and below an example for how solutions (Server Cluster, High performance hardware) contribute to values (Responsive Browsing, System Reliability) at some cost. By quantifying the values, a result can be calculated which yields a ratio of cost / performance. By discussing this with your team, you can check whether you can already generate *something* that already creates *some* value for your customer in the next cycle, and *lots of value* before the project ends!

Table 10.2 simplified
impact estimation table

Design Ideas -> Requirements	<u>Server Cluster</u>	<u>High-performance hardware</u>	Sum of Impact ^a
<u>Responsive Browsing</u> Baseline: 5 sec. Goal: 3 sec.			
Scale and % impact ^b	3 ± 1 sec. 100% ± 50	4 ± 1 sec. 50% ± 50	150% ± 100
Evidence and Credibility	CompetitorX has this configuration and response <- Jill Jones 0.2	Moon Microsystems has customers achieving this <- Moon Sys Eng. 0.1	
<u>System Reliability</u> Baseline: 3000 hours MTBF. Goal: 3500			
Scale and % impact	3200 ± 200. 40% ± 40	3100 ± 200. 20% ± 40	60% ± 80
Evidence and Credibility	CompetitorX has this config and "suspected" reliability <- Jill Jones 0.2	Moon Microsystems has customers achieving this <- Moon Sys Eng. 0.1	
Sum of Impact^c	140%	70%	
<u>Capital/Dev Cost</u> Baseline: \$0 USD. Budget: \$200K			
Amount and %	\$20K ± 10K. 10% ± 5	\$100K ± 10K. 50% ± 5	60% ± 10
Evidence and Credibility	Bob's friend guesses this cost on another project <- Bob Bones 0.1	Moon firm quote <- Moon Sales Rep. 1.0	
Benefit-to-Cost Ratio^c	14 (140% / 10%)	1.4 (70% / 50%)	
Impact Credibility Adjust Cost Credibility Adjust	0.84 (14 * .3 * .2) ^d 0.08 (0.84 * .1)	0.01 (1.4 * .1 * .1) 0.01 (0.01 * 1.0)	

- Sum of impacts on a requirement may or may not be cumulative.
- The % impacts are with respect to the baseline.
- The sum of impacts of one design idea may or may not be cumulative. The total may or may not work as an estimate for comparison.
- Multiplying probabilities is a heuristic to reduce total to a reasonable magnitude.

Figure 19 Example Impact Estimation Table. Source: [3]

For more background information, I recommend to read the very well written booklets by Niels Malotau [2].

5.2.2.2 References

- Gilb, Tom, and Lindsey Brodie. *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Oxford Burlington, MA: Butterworth-Heinemann, 2005.

2. "N R Malotau - Consultancy: Booklets / Downloads." Accessed December 23, 2021. <https://www.malotau.eu/index.php?id=downloads>.
3. Larman, Craig. *Agile and Iterative Development: A Manager's Guide*. Agile Software Development Series. Boston, Mass.: Addison-Wesley, 2004.
4. Malotau, Niels. *Evolutionary Project Management Methods*. 1.6., n.d. <https://www.malotau.eu/doc.php?id=2>.

5.2.3 Rapid Learning Cycles

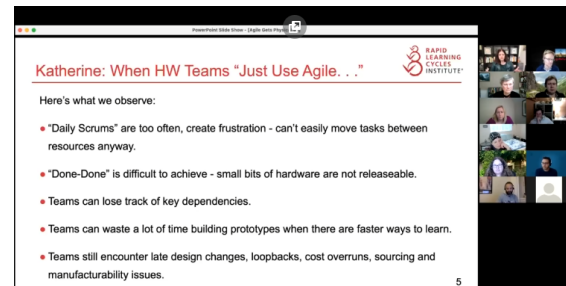
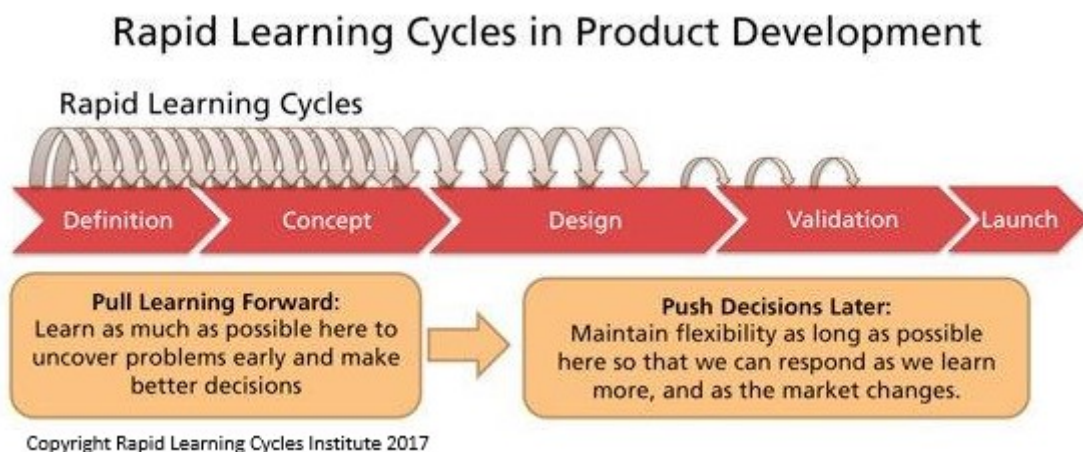


Figure 20 Still from "When Agile Gets Physical" talk. See[2]

Rapid Learning Cycles (RLC) was developed by Katherine Radeka, and is described in her book "The shortest distance between you and your new product"[1]. Several companies in the Twente region are using Rapid Learning Cycles as a development method.

The method originated from trying to adapt SCRUM to hardware development, which failed[2]. After some iterations, a method was found that tries to pull learning to the first part of the project, and tries to push decisions to later stages of the project, thereby taking time to make mistakes while you still can, and use the lessons learned to choose good implementations at a later moment. In current practice, we often see the reverse; a solution is chosen too soon, implemented, and while the full product is tested, a lot is learned about why this was not the good choice, and expensive respins have to be made.



5.2.3.1 Way of thinking

In RLC, you start off with a Core Hypothesis on your new product idea (e.g.: companies need a better robot to palletize small packages). From that Core Hypothesis, you derive Key Decisions. These are the decisions that have very high impact on your product, and which have great unknowns. To fill up the unknowns, you describe your knowledge gaps; what is needed to know to make a Key Decision? To fill those knowledge gaps, you do activities:

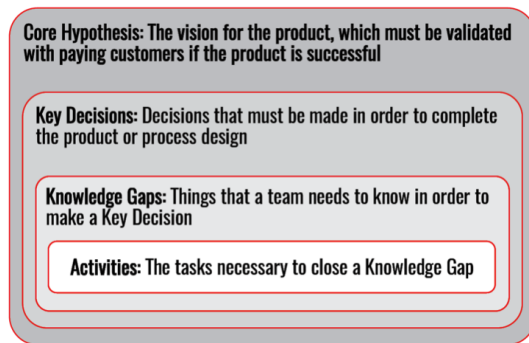


Figure 3.1: The Elements of the Rapid Learning Cycles Framework

Figure 21 Elements of RLC. Source: [1]



Figure 6.1: Not Every Decision Is a Key Decision

Figure 22 Not every decision is a key decision. Source:[1]

Please note, that Key decisions are not only derived from **technical** issues, but also from the **customer** and **business model** issues! For instance, knowing whether something is sold as a one-off product (MRI-scanner) or as a service ("MRI scan as a service", pay per scan) greatly impacts design choices.

Before and during the project, the team monitors what key decisions are still open, and how to get enough knowledge to close them. A planning board is maintained to show the progress. At so-called "Integration Events", multiple Key Decisions can be made in the presence of multiple stakeholders.

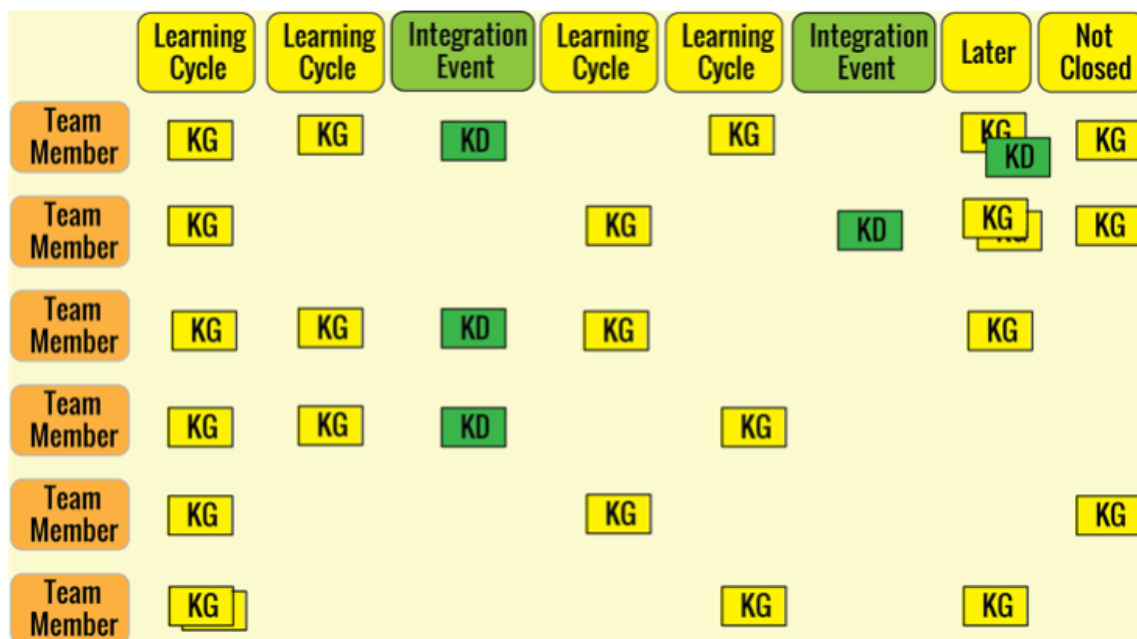


Figure 8.1: The Learning Cycles Plan

Figure 23 Learning Cycles Plan. KG=Knowledge Gap, KD=Key Decision. Source: [1]

5.2.3.1.1 Documentation, shareability

One of the good features of RLC is that documentation is lightweight and highly shareable. Each Knowledge Gap learning cycle is concluded with a Knowledge Gap report, which summarizes what is learned, and how. The Key Decisions are taken based on those reports, and generate their own Key Decision reports. In the default templates these are simple A4 / A3 documents. Not only does this help in the discussion with stakeholders, it also helps in sharing insights *across* projects. According to Scania[3] this really works:

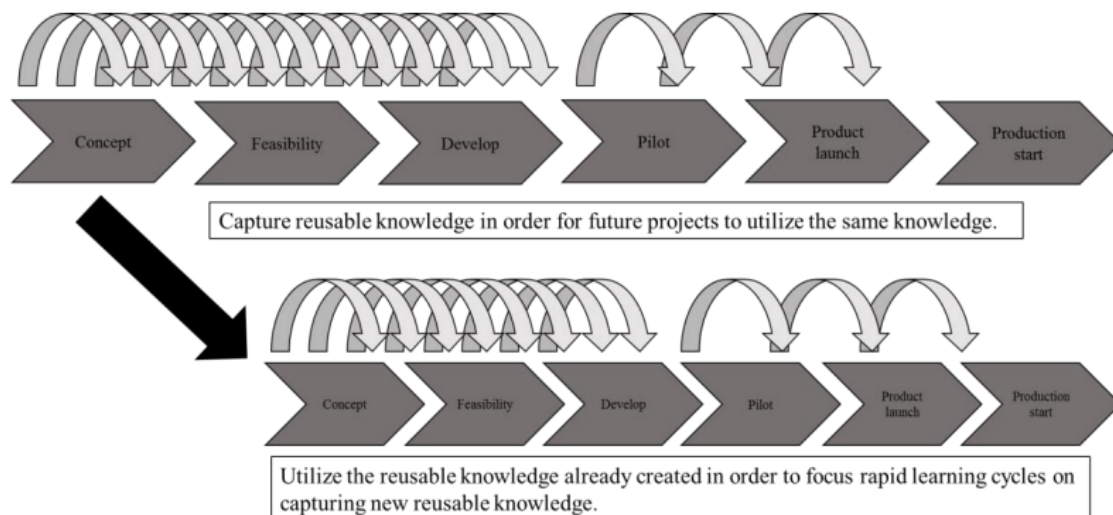


Figure 2.4-4 – Visualization of the leverages from implementation of RLC into product development projects, adapted from Radeka (2011). The leverages include the ability to carry out development projects in less time than before due to the knowledge which has already been created, which is re-used in future development projects.

Figure 24 Shortening development by reusing knowledge. Source: [3]

5.2.3.2 References

1. Radeka, Katherine. *The Shortest Distance between You and Your New Product: How Innovators Use Rapid Learning Cycles to Get Their Best Ideas to Market Faster*, 2017.
2. High Velocity Innovation. "Agile for Hardware: When Agile Gets Physical," April 19, 2021. <https://highvelocityinnovation.com/agile-for-hardware-when-agile-gets-physical/>.
3. Johansson, David, and Victor Persson. "Integrating Rapid Learning Cycles into Hardware Development - a Practical Improvement Project within Chassis Development at Scania CV AB." Chalmers University of Technology, 2016. <https://publications.lib.chalmers.se/records/fulltext/238022/238022.pdf>.

5.3 Product Line Engineering

Feature-based Product Line Engineering is a way of designing products by configuring them from existing features. A product is described with a "Bill of Features" that includes or excludes subsets of existing parts. Many large organizations have seen benefits of this approach, mostly in defense and automotive markets.

Although organizations clearly have lower costs due to the combined maintenance of common sections, it does take a management paradigm shift to think about how costs / benefits are shared over these common assets.

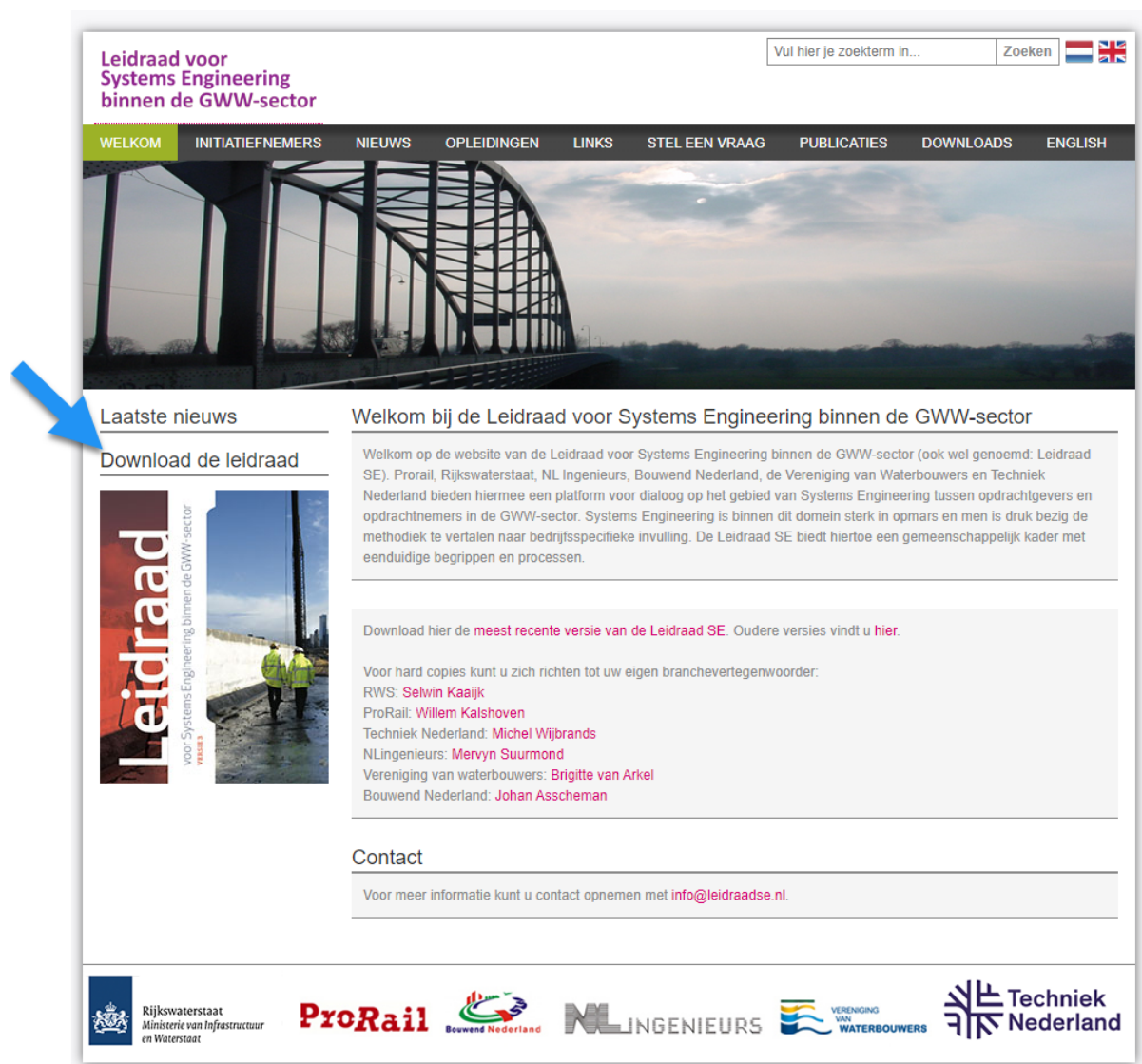
As with [Model Based Systems Engineering](#) (see page 26), applying PLE will need experts to use tooling to successfully use configured assets in your factory. The question is whether small and medium enterprises can come up with creative solutions to use the benefits of PLE without having to buy all the necessary tooling.

This chapter may need serious updating. For now I put some links to useful resources [here](#), if requested I can try to summarize the wealth of information here.

- Explanation on website: <http://www.productlineengineering.com/overview/what-is-ple.html>
- This paper was also published in an INCOSE magazine, and does a great job of showing in detail what choices have to be made to fully implement Product Line Engineering: [PLE_Industrial_Mainstream_Whitepaper_2020.pages \(biglever.com\)](http://www.biglever.com)²² \
- The INCOSE PLE primer. It is available for download from <https://www.incose.org/incose-member-resources/working-groups/analytic/product-lines>, although you might need to make an account for this. This Primer of less than 10 pages also does a nice job of showing not only the benefits, but also the organizational context that needs to be changed in order to effectively work with PLE.

5.4 A great additional resource

In civil engineering, stakes are high to “develop the right thing”, when thinking about tunnels, railroads or pipelines. From this sector, another “guideline” was made, that is very comprehensible and gives a lot of practical examples. This Leidraad SE was mentioned in the introduction of Systems Design and Systems Engineering, but deserves special attention here! Go to <http://www.leidraadse.nl> to download it.



The screenshot shows the website 'Leidraad voor Systems Engineering binnen de GWW-sector'. The header includes a search bar and navigation links: WELKOM, INITIATIEFNEMERS, NIEUWS, OPLEIDINGEN, LINKS, STEL EEN VRAAG, PUBLICATIES, DOWNLOADS, and ENGLISH. The main banner features a large image of a bridge. Below the banner, the 'Laatste nieuws' section highlights the 'Download de leidraad' link. The 'Welkom' section provides an overview of the website's purpose and lists contact persons for various organizations: RWS (Selwin Kaaijk), ProRail (Willem Kalshoven), Techniek Nederland (Michel Wijbrands), NLingenieurs (Mervyn Suurmond), Vereniging van waterbouwers (Brigitte van Arkel), and Bouwend Nederland (Johan Asscheman). The footer displays logos for the Rijkswaterstaat, ProRail, Bouwend Nederland, NLingenieurs, Vereniging van Waterbouwers, and Techniek Nederland.

²² https://biglever.com/wp-content/uploads/2018/11/PLE_Industrial_Mainstream_whitepaper.pdf