

Software Engineering in Industrial Practice (SEIP)

Dr. Ralf S. Engelschall

HI Minimize HARDWARE Idleness	DE Minimize DESIGN Excessiveness	HE Minimize HUMAN Effort
<p>Minimize the idleness and maximize the utilization of existing hardware resources.</p> <p>Rationale: Unused or under-utilized hardware are an unnecessary waste of already available resources.</p> <p>Keywords: Virtualization, Utilization.</p>	<p>Minimize the excessiveness and maximize the adequacy of solution designs.</p> <p>Rationale: Non-adequate designs cause unnecessary complexity and waste resources.</p> <p>Keywords: Reduced Libraries, Immutability.</p>	<p>Minimize the efforts of humans and maximize the efforts of machines in all production and operation processes.</p> <p>Rationale: Delegating tasks to machines gives humans the possibility to concentrate on more important tasks.</p> <p>Keywords: Computer, Robot, Automation.</p>
SI Minimize SOFTWARE Inefficiency	SE Minimize SOLUTION Ephemerality	EC Minimize ENERGY Consumption
<p>Minimize the inefficiency and maximize the efficiency of software applications and their development processes.</p> <p>Rationale: Efficient software and development processes consume less resources.</p> <p>Keywords: Caching, Monolith.</p>	<p>Minimize the ephemerality and maximize the life-span of any type of solutions.</p> <p>Rationale: Short life-spans of solutions cause unnecessary short renewals and this way wastes resources.</p> <p>Keywords: High Quality, Best Practice.</p>	<p>Minimize the consumption and maximize the saving of energy in all production and operation processes.</p> <p>Rationale: Electric energy still has to be partially generated from non-renewable resources.</p> <p>Keywords: Eco Mode, Reduced CI/CD.</p>
IA Minimize INFORMATION Amount	EE Minimize ECOSYSTEM Exploitation	CE Minimize CARBON Emission
<p>Minimize the total amount of gathered, transmitted, stored and spreaded information.</p> <p>Rationale: Reduced amount of information means less data transmission, less data storage, less GDPR issues, etc.</p> <p>Keywords: Compression, No Big Data.</p>	<p>Minimize the exploitation and maximize the back-contribution in any type of ecosystems.</p> <p>Rationale: The consumer and provider behaviour have to be in balance for every long-lasting ecosystem.</p> <p>Keywords: Open Source Software.</p>	<p>Minimize the carbon emission and hence the footprint during any type of production and operation processes.</p> <p>Rationale: Climate change and global warming is partially caused or at least accelerated by carbon emissions.</p> <p>Keywords: Reduced CO2 Footprint.</p>

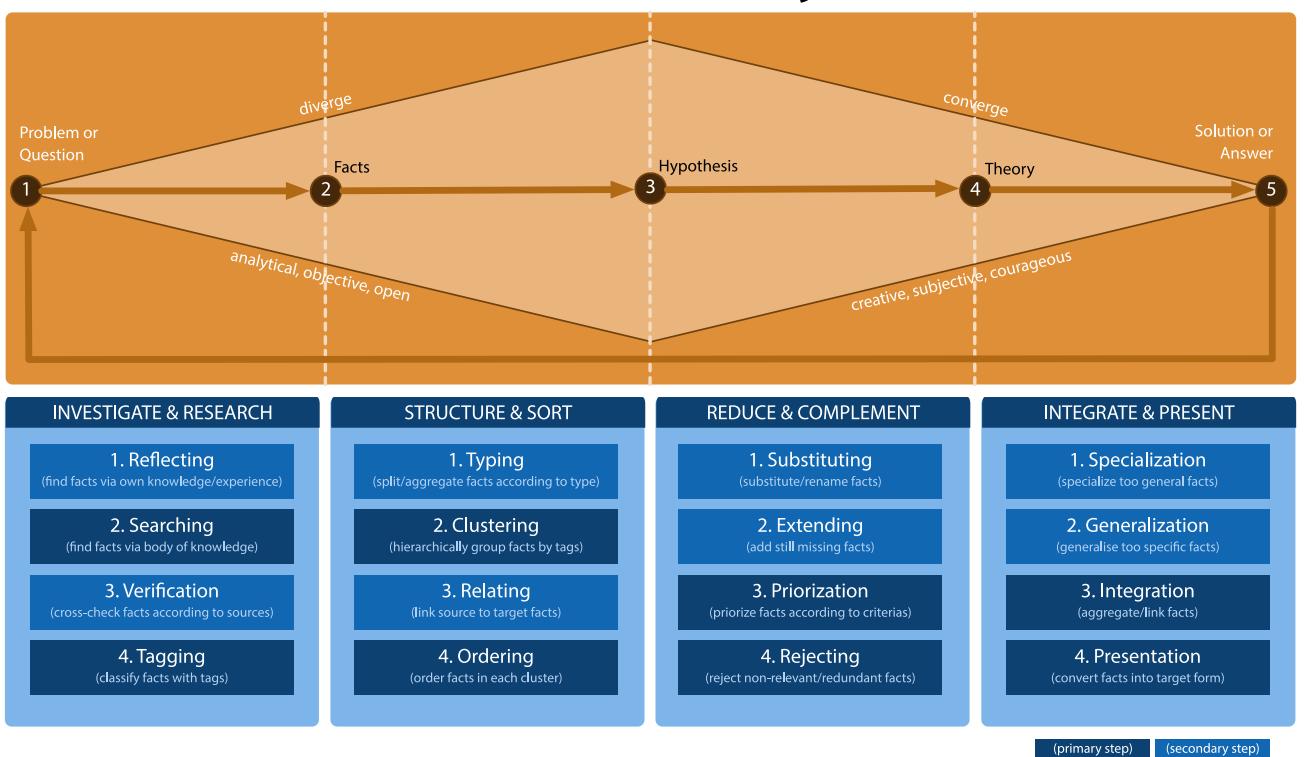
Sustainable action should be a matter of course, since there will always be others coming after us. In Software Engineering, the following minimization principles lend themselves to acting sustainably:

Minimize the idleness and maximize the utilization of existing hardware resources; Minimize the inefficiency and maximize the efficiency of software applications and their development processes; Minimize the total amount of gathered, transmitted, stored, and spreaded information; Minimize the excessiveness and maximize the adequacy of solution designs; Minimize the ephemerality and maximize the life span of any type of solutions; Minimize the exploitation and maximize the back-contribution in any type of ecosystems; Minimize the efforts of humans and maximize the efforts of machines in all production and operation processes; Minimize the consumption and maximize the saving of energy in all production and operation processes; and: Minimize the carbon emission and hence the footprint during any type of production and operation processes.

Questions

?

Is the Best Practice of Continuous Integration (CI) a sustainable way of acting?



The architect must regularly “think clearly” about certain problems or issues. For this purpose, it is a good idea to go through the four-stage **Think Clearly** process, once or even iteratively if required. The process consists of four clearly differentiated disciplines.

In the first two disciplines **Investigate & Research** and **Structure & Sort**, one tries to act analytically, objectively, and openly to **diverge** the problem or the question, i.e., to collect many facts and to build up a hypothesis by structuring and sorting.

In the last two disciplines **Reduce & Complement** and **Integrate & Present**, one tries to act creatively, subjectively, and courageously and to finally **converge** with regard to the problem or question, i.e., to reduce the hypothesis to a coherent theory and then integrate it to the solution or the answer.

In the first discipline **Investigate & Research** one finds facts about own knowledge and experience (**Reflecting**) or by research in external sources (**Searching**), ones verifies the facts through sources (**Verification**) and ones classifies the facts by enrichment with tags (**Tagging**).

In the second discipline **Structure & Sort** one divides or aggregates the facts for the best possible “sort purity” (**Typing**), one groups the facts hierarchically via tags (**Clustering**), one connects related facts across groups (**Relating**) and one sorts the facts in each group.

In the third discipline, **Reduce & Complement**, one replaces facts or renames facts if necessary (**Substituting**), adds missing facts (**Extending**), prioritizes the facts according to certain criteria (**Prioritization**) and discard irrelevant or redundant facts (**Rejecting**).

In the fourth and last discipline **Integrate & Present** one specializes too general facts if necessary (**Specialization**), generalizes too specific facts if necessary (**Generalization**), one aggregates or combines facts (**Integration**) and converts facts into their target representation (**Presentation**).

Questions

- ?
- In the **Think Clearly** process to “think clearly,” one should think how about the first two and the last two disciplines?

Research	RE	Abstraction	AB	Lateral Thinking	LT	Backward Search	BS
Crawling the problem domain's body of knowledge to find starting points.		Solving the problem in a model of the problem before applying it to the real problem to get a better understanding.		Approaching the problem indirectly and creatively to find a not obvious solving lever.		Looking at the expected results and determine which operations could bring you to them.	
Brainstorming	BR	Generalization	GE	Hypothesis Proof	HP	Backtracking	BT
Suggesting larger number of solution ideas for further combination and development.		Thinking about the problem more abstract to get rid of special cases.		Assuming a possible solution and trying to prove (or disprove) the assumption to find starting points.		Remembering path towards the solution and on failure tracking back and choosing a new path.	
Analogy	AN	Specialization	SP	Root Cause	RC	Divide & Conquer	DC
Thinking in terms of similar problems for which solutions are known to get inspired.		Solving a special case first to get an impression towards the full solution.		Asking "Why?" five times in sequence to explore the cause-and-effect relationships underlying the problem.		Breaking down the large complex problem into smaller, easier solvable partial problems.	
Reduction	RD	Variation	VA	Means End	ME	Trial & Error	TE
Transform the problem into another one for which a solutions already exists to reduce solving efforts.		Changing the problem context or expressing the problem differently to find a not obvious solving lever.		Choosing an action from scratch just at each step to move closer and closer to the solution.		As a last resort, brute-force testing all potential solutions in case of a small enough total solution space.	

Definition: **Heuristic** — fallible experience-based technique or strategy for problem solving in case *Rule of Thumb Guessing*, *Intuitive Judgement*, *Common Sense* and *Stereotyping* are either not sufficient or not appropriate.

The **Problem Solving Heuristics** are experience-based techniques or strategies that can be used for problem-solving when other approaches do not make any progress.

The heuristics are mainly used for inspiration so that you don't spend too long and instead find a new starting point for solving the problem ("If you find yourself in a hole, stop digging!").

With **Research** one searches for facts, with **Brainstorming** ones proposes a large number of spontaneous solution ideas, in **Analogy** one thinks of similar problems that have already been solved, and in **Reduction** one transforms the problem into an already solved problem.

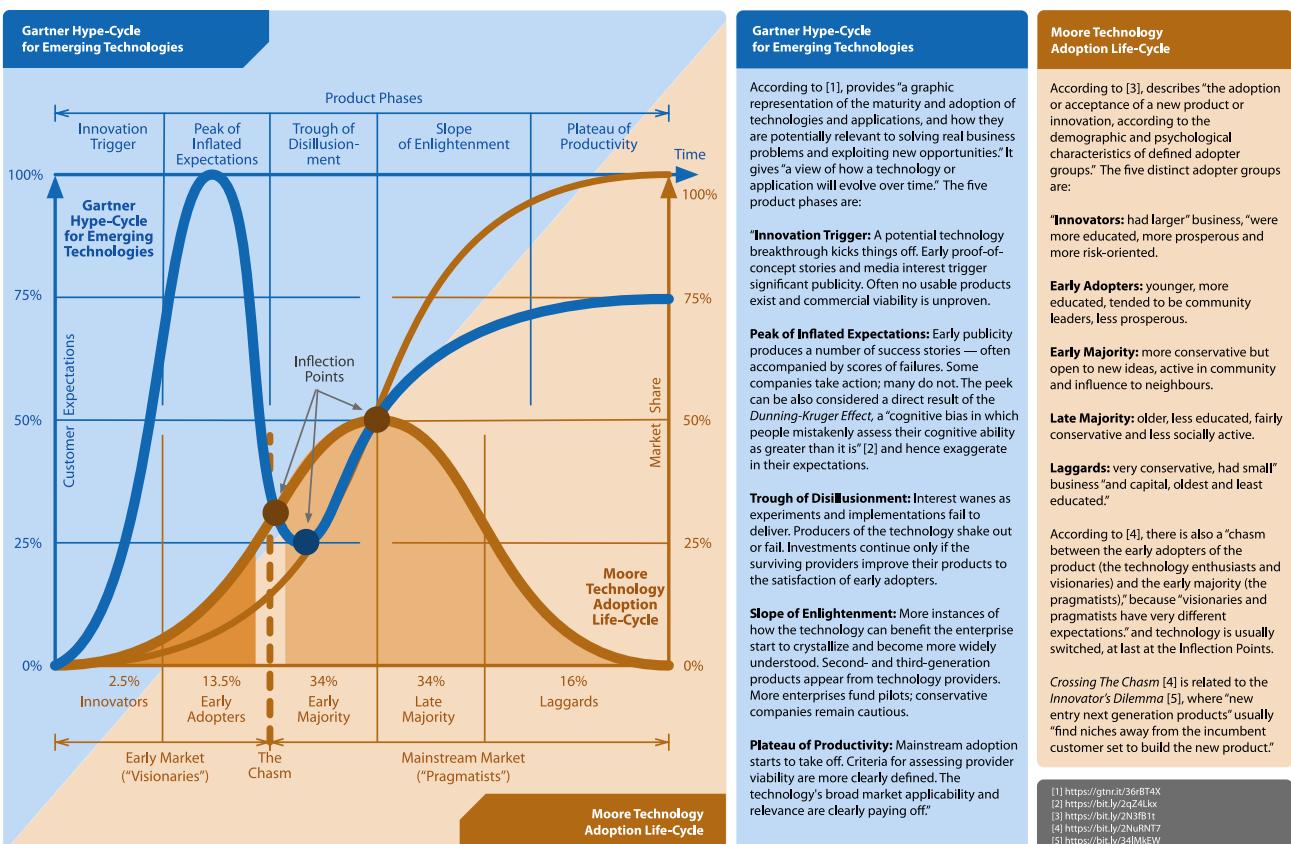
With **Abstraction** one solves the problem first in a more abstract model, with **Generalization** one generalizes the problem to one with fewer special cases, in **Specialization** one tries to be inspired by a special case, and in **Variation** one tries to change one's perspective on the problem.

In **Lateral Thinking** one approaches the problem in a deliberately indirect and creative way, with **Hypothesis Proof** one searches for a solution by proof for a possible or not possible fictitious solution, with **Root Cause** one goes to the root of the problem step by step and in **Means End** one tries to approach the solution in small steps.

In **Backward Search** one tries to get from a fictitious solution backward on the way to the solution, with **Backtracking** one chooses a partly new path in the direction of the solution in case of a failure, in **Divide & Conquer** one breaks down the big problem into smaller and easier to solve subproblems, and with **Trial & Error** one tries all solution combinations as a last resort and/or if the solution space is small enough.

Questions

- ?
- When is the rather mundane **Problem Solving Heuristic** called **Trial & Error** acceptable?



One can classify a technology (or a concrete technological product) very well using two models, the **Technology Life-Cycles**: the **Gartner Hype-Cycle for Emerging Technologies** and the **Moore Technology Adoption Life-Cycle**.

The **Gartner Hype-Cycle for Emerging Technologies** shows the usual life-cycle of a technology over the temporal **Product Phases** (in the x-axis) **Innovation Trigger**, **Peak of Inflated Expectations**, **Trough of Disillusionment**, **Slope of Enlightenment** and **Plateau of Productivity**, and via **Customer Expectations** (in the y-axis). It thus primarily maps the maturity level of a technology and shows expectations of the technology on the market at the time.

The **Moore Technology Adoption Life-Cycle** shows the level of acceptance of the technology in different types of markets. These markets are characterized by the fundamentally different market participants **Innovators**, **Early Adopters**, **Early Majority**, **Late Majority** and **Laggards**, where usually a certain gap (**The Chasm**) exists between the **Early Market** of visionaries and the **Mainstream Market** of pragmatists. To bridge this gap, a technology usually has to be developed in a second generation.

The **Moore Technology Adoption Life-Cycle** is also related to the **Innovators Dilemma**. This is because it is possible to calculate the maximum achievable market share of a technology over time in the form of an S-curve. The key points for a technology are at about 25% (**The Chasm**) and 50% market share. In addition, this S-curve shows the **Innovators Dilemma**, i.e., the fact that a new technology always has to bridge a dry spell in niches of the **Early Market** before it can achieve a larger share of the **Mainstream Market**.

Questions

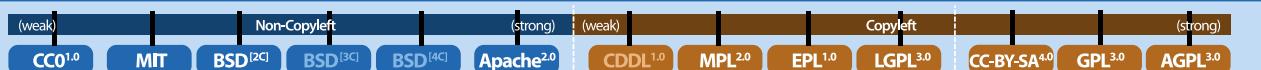
- ?
- Which model of a **Technology Life-Cycle** represents the maturity of a technology over time?
- ?
- Which model of a **Technology Life-Cycle** represents the adoption of a technology in different markets?

Open Source Definition

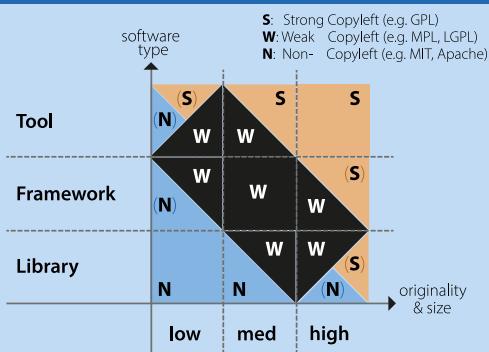
Distribution terms (license) of Open Source Software must be compliant with the following criterias:

- Free Redistribution
 - (Original) Source Code (Availability)
 - Derived Works (Allowance)
 - Integrity of the Author's Source Code**
 - No Discrimination Against Persons or Groups**
 - No Discrimination Against Fields of Endeavor**
 - Distribution of (Non-Exclusive) License**
 - License Must Not Be Specific to a Product
 - License Must Not Restrict Other Software
 - License Must Be Technology-Neutral

Most Popular Open Source Licenses



Choosing an Open Source License



Open Source Personality Streams

§ Software Sharing

Dogmatism
Social Equity
Politics

@ Software Hacking

Fundamentalism
Art
Hacking

€ Software Engineering

Pragmatism
Business
Engineering

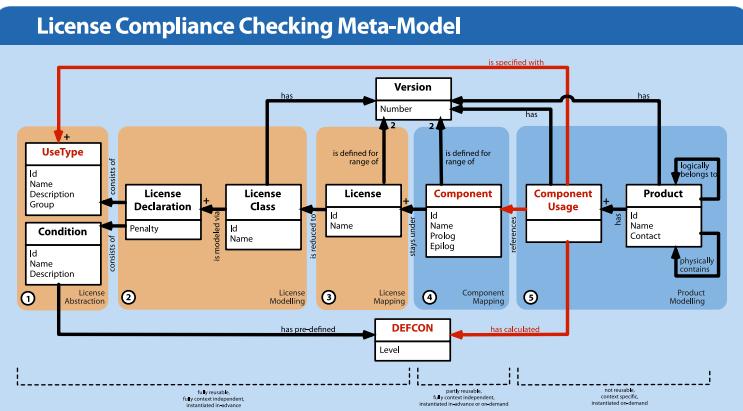


Open Source Software is software that has been placed under an **Open Source License**. All licenses recognized as **Open Source Licenses** meet the **Open Source Definition**, which states, among other things, that the software may be freely distributed in source code and changed and that there is no discrimination of persons, groups, or purposes.

In practice, one knows three **Open Source Personality Streams**: **Software Sharing** with dogmatic and political persons, who are fighting for social justice; **Software Hacking** with fundamental and artistic persons, who develop software with maximum ambition; and **Software Engineering** with pragmatic people who use the software in practice.

There are hundreds of **Open Source Licenses**. However, one can split them into a few classes and sort them according to their strength, i.e., how strongly they protect the software itself. One distinguishes generally between licenses without and with a so-called **Copyleft** effect. This consists of license clauses in order to keep the original software free (in the sense of freedom and availability, not free of charge) and additionally to keep all modifications and extensions to the software also free.

The weakest license in practice is **Creative Commons Zero (CC0)** (or **Public Domain**), which effectively allows anyone to do anything.

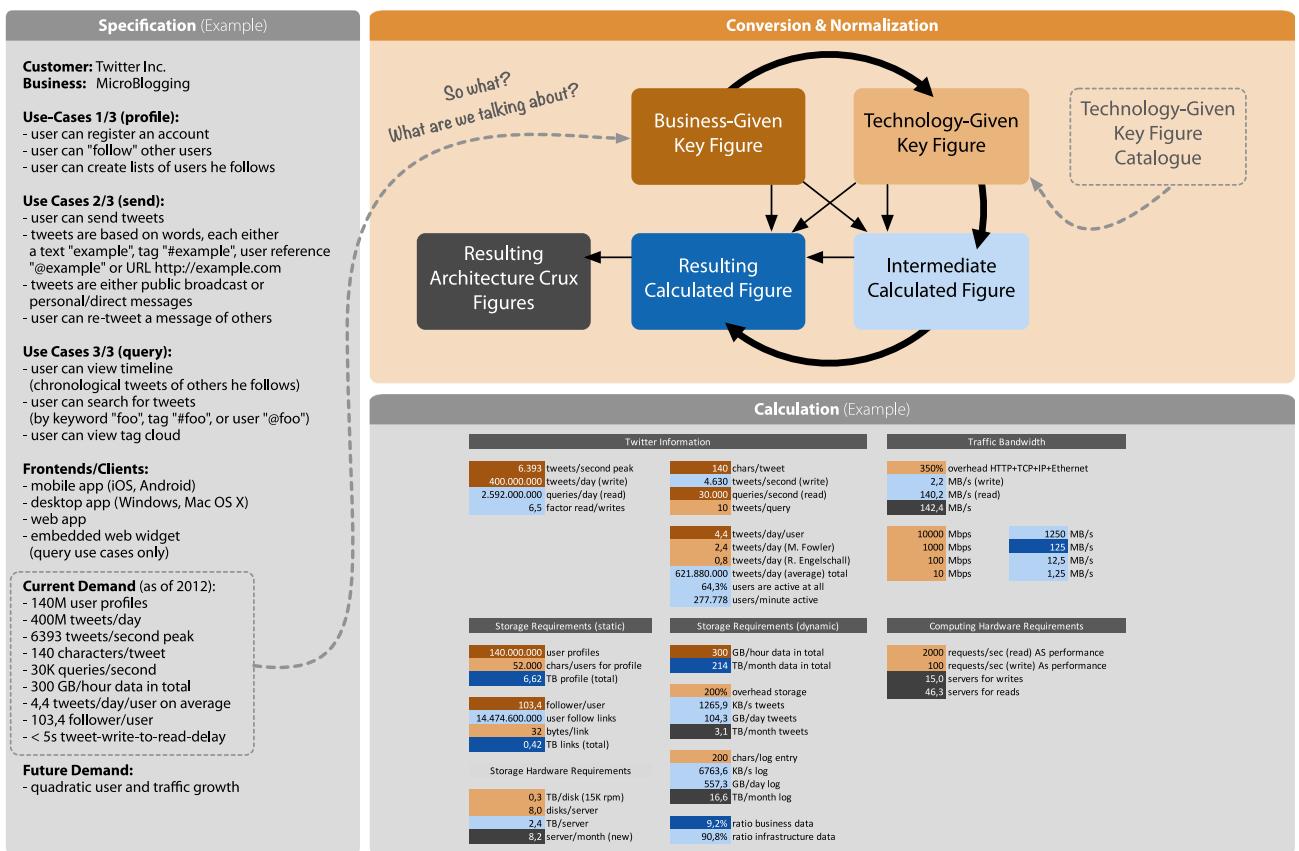


The strongest license is the **Affero General Public License (AGPL)**, which protects software even in the case of use in the form of Software as a Service (SaaS). At the Copyleft boundary is the **Apache License**, which does not yet have a Copyleft effect but still tries to maximally protect the software and the originator.

In practice, a distinction is made between licenses with no, weak and strong Copyleft. To decide for a software under which class of license one publishes it, one differentiates between two dimensions: on the one hand, the type of software (**Tool**, **Framework** or **Library**) and on the other hand, the level of creation of the software. A **Tool** or a **Framework** with a medium or high level of creation is usually under weak or even strong Copyleft to protect the software and the author to the maximum. A **Library** or a **Framework** with a medium or low level of intellectual property is under a weak or even no Copyleft in order to achieve a maximum distribution of the software.

Questions

- ?) What do you call the effect in licenses of **Open Source Software**, in which the software remains free (in the sense of freedom and availability, not in the sense of free of charge) and additionally all modifications and extensions remain free as well?



To get a better "feeling" for the scope and the degree of difficulty of an architecture to be developed, it is a good idea to do a **Back of the Envelope Calculation** ("rough calculation"). The method is as follows: in a spreadsheet, a two-column table is created in which the first column contains the number and the second column contains the unit.

Now, in the first step, the **Business-Given Key Figures**, i.e., the technically known numbers, are entered into the table as the first rows. They get the first color for differentiation.

Because these numbers usually do not tell enough, in a second step, different **Technology-Given Key Figures** are entered as rows. These may be taken from existing catalogs or are available from your own experience. They are given the second color for differentiation and serve above all as comparative figures to the **Business-Given Key Figures**.

In the third step, both the **Business-Given Key Figures** and the **Technology-Given Key Figures** and compared to each other. The intermediate results, called **Intermediate Calculated Figures**, are spreadsheet cells with formulas, which get the third color to distinguish them.

Whenever an **Intermediate Calculated Figure** (or possibly already an **Business-Given Key Figure** or a **Technology-Given Key Figure**) provides a decisive hint or insight, you change the row to the fourth color. If this insight has potential relevance for the subsequent architecture (and thus represents a key point), the row is changed to the fifth color, which shows the **Resulting Architecture Crux Figure**.

Afterwards, you can optionally bundle the different rows into logical groups in the spreadsheet to make the spreadsheet clearer.

Questions

- ② What method can be used to get a better "feel" for the scope and difficulty of an architecture to be developed?

Weighted Decision Matrix

		A ₁	A ₂	...	A _n
C ₁	w ₁	E _{1,1}	E _{1,2}	...	E _{1,n}
C ₂	w ₂	E _{2,1}	E _{2,2}	...	E _{2,n}
...
C _m	w _m	E _{m,1}	E _{m,2}	...	E _{m,n}
		R ₁	R ₂	...	R _n

Light-Weight Alternative:
qualitatively cherry-picking major positive/negative backlog criteria

		A ₁	A ₂	...	A _n
+	C ₁ C ₄ C ₈	C ₁ C ₂ C ₇	...	C ₂ C ₉ C ₈	
-	C ₂ C ₃ C ₄	C ₇ C ₅	
	Decision for A _{best}				

Standard Criteria Catalogs

Software Selection:

- + Suitable Functionality
- + Available Usage Examples
- + Reasonable Documentation
- + Reasonable Support
- + Permissive License
- + Long-Term Release Track Record
- + Current Market Momentum

Software Selection (Open Source):

- + Clean Source Code
- + Clean Build Process
- + Open Source License

Software Selection (Library):

- + Non-Invasive Programming Model
- + Orthogonal Application Programming Interface
- + Minimum/No Dependencies
- + Non-Copyleft Open Source License

Software Selection (Framework):

- + Orthogonal Application Programming Interface
- + Adequate Dependencies
- + Non-Overlapping Scope
- + Non-Copyleft Open Source License

Software Selection (Tool):

- + Clean Deployment Procedure
- + Pleasant Command-Line Interface

Software Selection (Application):

- + Clean Deployment Procedure
- + Pleasant Graphical User Interface

Software Architecture Evaluation:

- + Meets Functional Requirements
- + Meets Non-Functional Requirements
- + Adequate Technology Overhead
- + Single Dependency Direction
- + Distance to State of the Art ("modern")
- + Distance to Most Simple Approach ("adequate")
- + Distance to Mainstream Approach ("mainstream")
- + Documented Architecture Decisions ("rationales")
- + Documented Architecture Views
- + Documented Architecture Perspectives (NFR)

Notice
It's about **subjective decision transparency**, not about **objective decision making!**

Decision Making Process

```

graph TD
    Start(( )) --> EstablishCriterias[Establish Criterias]
    EstablishCriterias --> OrganizeHierarchically{Organize Criterias Hierarchically}
    OrganizeHierarchically --> WeightCriteria[Weight Criterias (simply or AHP)]
    WeightCriteria --> FindAlternatives[Find Alternatives]
    FindAlternatives --> SortAlternatives{Sort Alternatives Sequentially}
    SortAlternatives --> EvaluateAlternatives[Evaluate Alternatives against Criterias]
    EvaluateAlternatives --> DetermineRanking[Determine Alternatives Ranking]
    DetermineRanking --> End((( )))
    WeightCriteria --> SortAlternatives
    EvaluateAlternatives --> DetermineRanking
  
```

In order to make qualitative decisions transparently and comprehensibly (but not necessarily objective) and at the same time to document the decision-making process, one can apply the method of the **Weighted Decision Matrix**.

The prerequisite is that the decision to be made is the choice of one of many alternatives. These are entered in a spreadsheet as columns. Optionally, the alternatives can be put into a meaningful order.

Then one specifies different criteria, which are to be used for the decision. The goal is to distinguish the alternatives with as less criteria as possible. Each criterion is given a weighting. Optionally, the criteria can be grouped into a hierarchy and the groups into a meaningful order.

Subsequently, one determines a weighting for each criterion, which indicates how strongly the criterion is considered in the decision-making process.

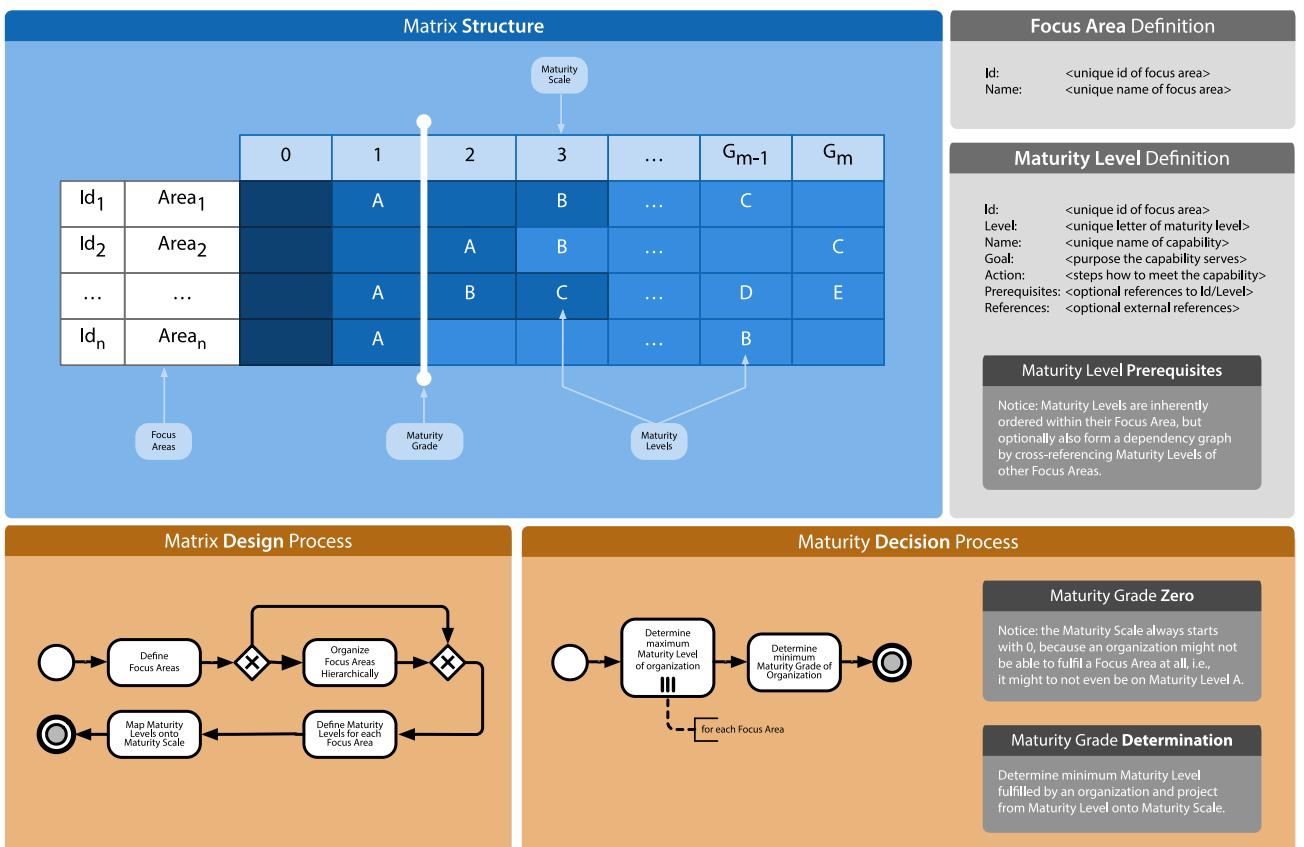
Now all alternatives are evaluated against all criteria. With few criteria and many alternatives, one can evaluate all criteria per alternative. If there are many criteria, it is advisable to evaluate all alternatives per criterion. As an evaluation scale, it is advisable to use -2, -1, 0, +1, +2, in order to have a middle (0), positive/negative evaluations (+1, -1) and positive/negative superlatives (+2, -2).

Finally, for each alternative, the product sum of the criterion weighting column and the alternative rating column is calculated. The decision is then made for the alternative with the maximum amount in the product sum.

Questions

- ❓ How to make the qualitative decision to choose one of many alternatives can be made transparently and comprehensibly and document it at the same time?

Focus Area Maturity Model



The Focus Area Maturity Model (FAMM) is a method to assess the maturity of an organization with respect to a specific topic area.

The structure of a FAMM is a matrix of horizontal Focus Areas and their Maturity Levels and possible vertical Maturity Grades on a Maturity Scale. Per Focus Area there can be one or any number of Maturity Levels and their positions on the Maturity Scale are based on the importance of the Focus Areas and the relationship between the Focus Areas and their Maturity Levels. This matrix is designed in a first step for a topic area and is then fixed.

In order to determine the Maturity Level of an organization, determine for each Focus Area the maximum Maturity Level the organization fulfills. The Maturity Level of the organization is then derived from the minimum Maturity Level across all Focus Areas and the projection of this Maturity Level onto the Maturity Scale.

Since the Maturity Levels should only ever be positioned in the matrix above Maturity Grade 0, in the worst case, an organization has a Maturity Grade 0 if it does not fulfill a Focus Area at all.

Questions

- From whom can you determine the maturity level with the Focus Area Maturity Model (FAMM)?