TECHNISCHE UNIVERSITÄT MÜNCHEN

Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall

Think Clearly

INVESTIGATE & RESEARCH	STRUCTURE & SORT	REDUCE & COMPLEMENT	INTEGRATE & PRESENT
1. Reflecting (find facts via own knowledge/experience)	**1. Typing** (split/aggregate facts according to type)	**1. Substituting** (substitute/rename facts)	**1. Specialization** (specialize too general facts)
2. Searching (find facts via body of knowledge)	**2. Clustering** (hierarchically group facts by tags)	**2. Extending** (add still missing facts)	**2. Generalization** (generalise too specific facts)
3. Verification (cross-check facts according to sources)	**3. Relating** (link source to target facts)	**3. Priorization** (priorize facts according to criterias)	**3. Integration** (aggregate/link facts)
4. Tagging (classify facts with tags)	**4. Ordering** (order facts in each cluster)	**4. Rejecting** (reject non-relevant/redundant facts)	**4. Presentation** (convert facts into target form)

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 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?

Problem Solving Heuristics

Research RE — Crawling the problem domain's body of knowledge to find starting points.	**Abstraction** AB — Solving the problem in a model of the problem before applying it to the real problem to get a better understanding.	**Lateral Thinking** LT — Approaching the problem indirectly and creatively to find a not obvious solving lever.	**Backward Search** BS — Looking at the expected results and determine which operations could bring you to them.
Brainstorming BR — Suggesting larger number of solution ideas for further combination and development.	**Generalization** GE — Thinking about the problem more abstract to get rid of special cases.	**Hypothesis Proof** HP — Assuming a possible solution and trying to prove (or disprove) the assumption to find starting points. Q.E.D.	**Backtracking** BT — Remembering path towards the solution and on failure tracking back and choosing a new path.
Analogy AN — Thinking in terms of similar problems for which solutions are known to get inspired.	**Specialization** SP — Solving a special case first to get an impression towards the full solution.	**Root Cause** RC — Asking "Why?" five times in sequence to explore the cause-and-effect relationships underlying the problem. 5x WHY?	**Divide & Conquer** DC — Breaking down the large complex problem into smaller, easier solvable partial problems.
Reduction RD — Transform the problem into another one for which a solutions already exists to reduce solving efforts.	**Variation** VA — Changing the problem context or expressing the problem differently to find a not obvious solving lever.	**Means End** ME — Choosing an action from scratch just at each step to move closer and closer to the solution.	**Trial & Error** TE — 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.
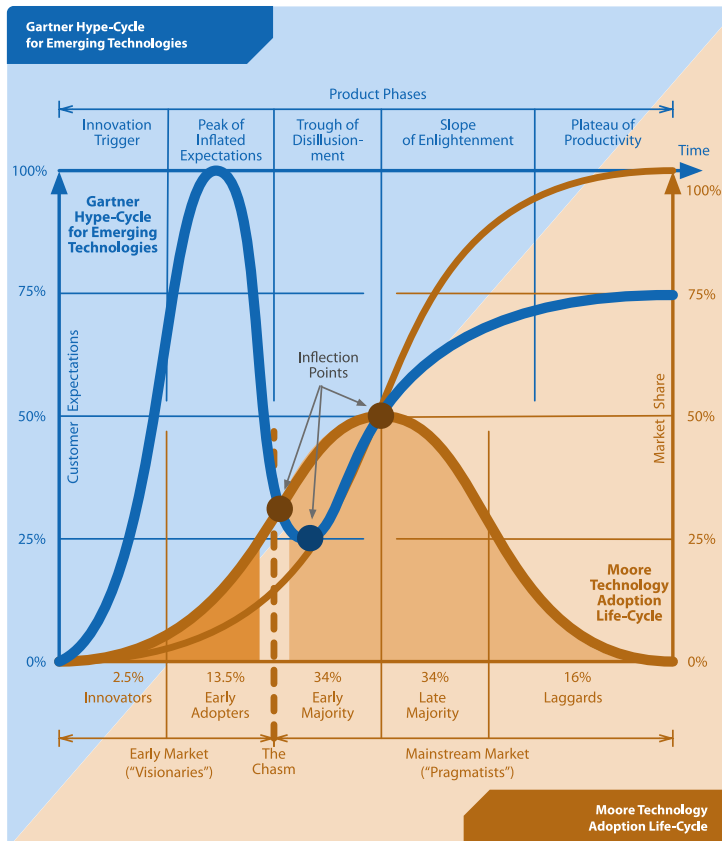
With **Abstraction** one solves the problem first in a more abstract model, with **Generalization** one generalizes the problem to one with fewer special cases, in **Specialization** one tries to be inspired by a special case, and in **Variation** one tries to change one's perspective on the problem.

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

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

Technology Life-Cycles

Gartner Hype-Cycle for Emerging Technologies

Product Phases

| Innovation Trigger | Peak of Inflated Expectations | Trough of Disillusion-ment | Slope of Enlightenment | Plateau of Productivity |

Time

Gartner Hype-Cycle for Emerging Technologies

Customer Expectations

100% / 75% / 50% / 25% / 0%

100% / 75% / 50% / 25% / 0%

Inflection Points

Market Share

Moore Technology Adoption Life-Cycle

| 2.5% Innovators | 13.5% Early Adopters | 34% Early Majority | 34% Late Majority | 16% Laggards |

Early Market ("Visionaries") | The Chasm | Mainstream Market ("Pragmatists")

Moore Technology Adoption Life-Cycle

Gartner Hype-Cycle for Emerging Technologies

According to [1], provides "a graphic representation of the maturity and adoption of technologies and applications, and how they are potentially relevant to solving real business problems and exploiting new opportunities." It gives "a view of how a technology or application will evolve over time." The five product phases are:

"**Innovation Trigger:** A potential technology breakthrough kicks things off. Early proof-of-concept stories and media interest trigger significant publicity. Often no usable products exist and commercial viability is unproven.

Peak of Inflated Expectations: Early publicity produces a number of success stories — often accompanied by scores of failures. Some companies take action; many do not. The peak can be also considered a direct result of the *Dunning-Kruger Effect*, a "cognitive bias in which people mistakenly assess their cognitive ability as greater than it is" [2] and hence exaggerate in their expectations.

Trough of Disillusionment: Interest wanes as experiments and implementations fail to deliver. Producers of the technology shake out or fail. Investments continue only if the surviving providers improve their products to the satisfaction of early adopters.

Slope of Enlightenment: More instances of how the technology can benefit the enterprise start to crystallize and become more widely understood. Second- and third-generation products appear from technology providers. More enterprises fund pilots; conservative companies remain cautious.

Plateau of Productivity: Mainstream adoption starts to take off. Criteria for assessing provider viability are more clearly defined. The technology's broad market applicability and relevance are clearly paying off."

Moore Technology Adoption Life-Cycle

According to [3], describes "the adoption or acceptance of a new product or innovation, according to the demographic and psychological characteristics of defined adopter groups." The five distinct adopter groups are:

"**Innovators:** had larger" business, "were more educated, more prosperous and more risk-oriented.

Early Adopters: younger, more educated, tended to be community leaders, less prosperous.

Early Majority: more conservative but open to new ideas, active in community and influence to neighbours.

Late Majority: older, less educated, fairly conservative and less socially active.

Laggards: very conservative, had small" business "and capital, oldest and least educated."

According to [4], there is also a "chasm between the early adopters of the product (the technology enthusiasts and visionaries) and the early majority (the pragmatists)," because "visionaries and pragmatists have very different expectations." and technology is usually switched, at last at the Inflection Points.

Crossing The Chasm [4] is related to the *Innovator's Dilemma* [5], where "new entry next generation products" usually "find niches away from the incumbent customer set to build the new product."

[1] https://gtnr.it/36rBT4X
[2] https://bit.ly/2qZ4Lkx
[3] https://bit.ly/2N3fB1t
[4] https://bit.ly/2NuRNT7
[5] https://bit.ly/34lMkEW

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 Inflates 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?

ARCHITECTURE FUNDAMENTALS

Open Source Software

TECHNISCHE UNIVERSITÄT MÜNCHEN

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**

Open Source Personality Streams

§ **Software Sharing** — Dogmatism / Social Equity / Politics

@ **Software Hacking** — Fundamentalism / Art / Hacking
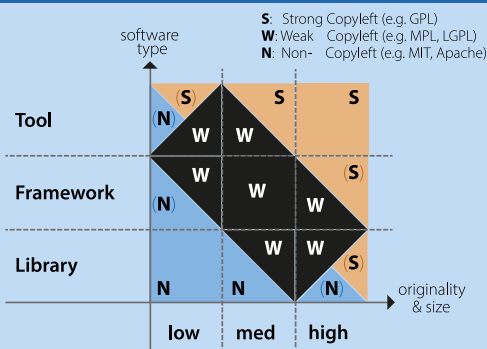
€ **Software Engineering** — Pragmatism / Business / Engineering

Science / Private / Industry

Most Popular Open Source Licenses

(weak) — Non-Copyleft — (strong) | (weak) — Copyleft — (strong)

CC0[1.0] | MIT | BSD[2C] | BSD[3C] | BSD[4C] | Apache[2.0] | CDDL[1.0] | MPL[2.0] | EPL[1.0] | LGPL[3.0] | CC-BY-SA[4.0] | GPL[3.0] | AGPL[3.0]

Choosing an Open Source License

S: Strong Copyleft (e.g. GPL)
W: Weak Copyleft (e.g. MPL, LGPL)
N: Non- Copyleft (e.g. MIT, Apache)

software type

Tool — (S) (N) W W S S
Framework — W W W (S) (N) W
Library — W W (S) N N (N)

originality & size — low / med / high

License Compliance Checking Meta-Model
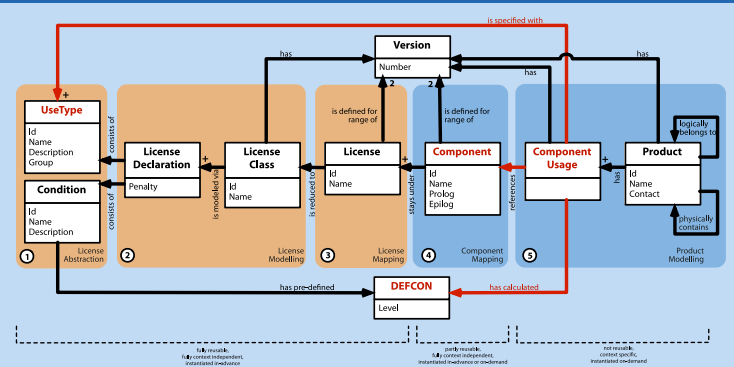
Open Source Software is software that has been placed under an **Open Source License**. All licenses recognized as **Open Source Licenses** meet the **Open Source Definition**, which states, among other things, that the software may be freely distributed in source code and changed and that there is no 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.
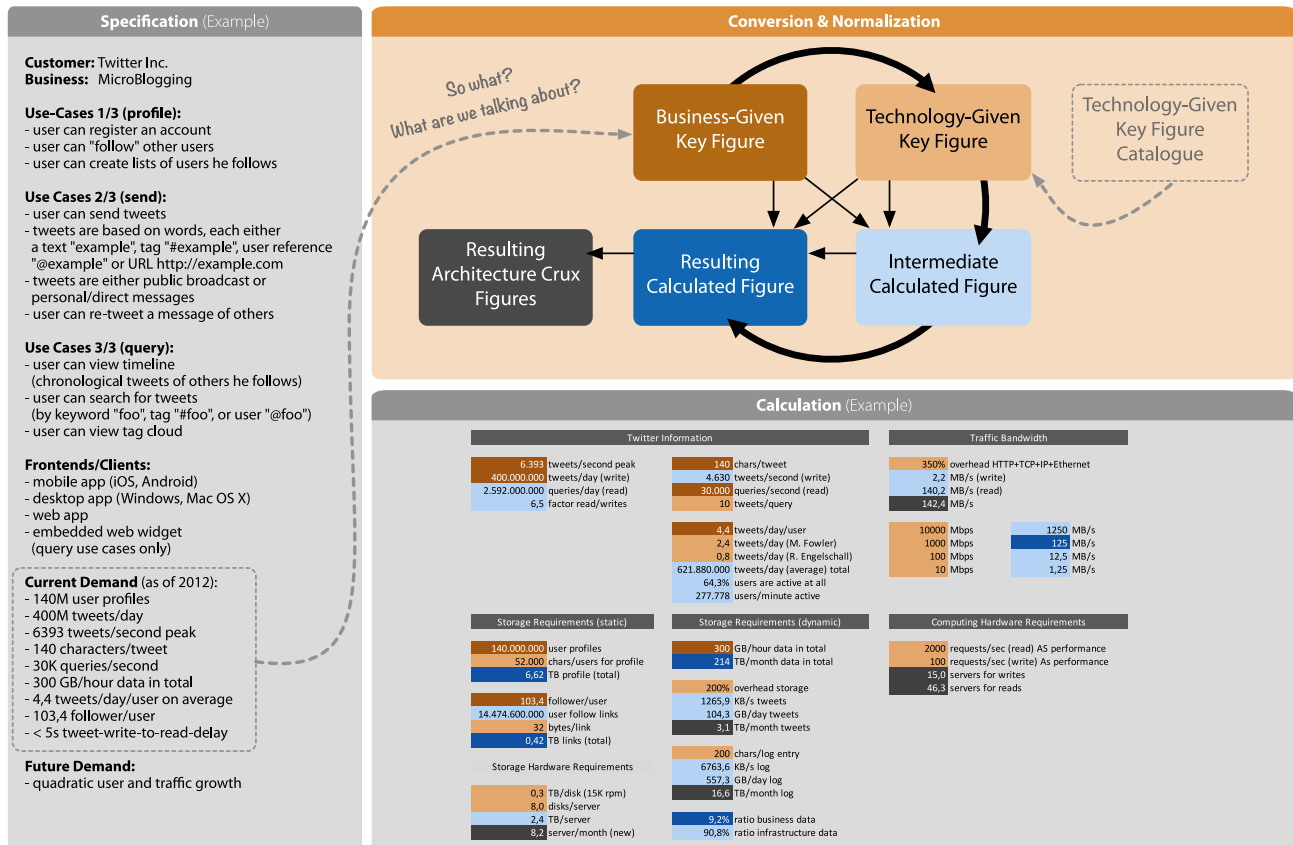
The weakest license in practice is **Creative Commons Zero (CC0)** (or **Public Domain**), which effectively allows anyone to do anything.

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 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?

Back of the Envelope Calculation

Specification (Example)

Customer: Twitter Inc.
Business: MicroBlogging

Use-Cases 1/3 (profile):
- user can register an account
- user can "follow" other users
- user can create lists of users he follows

Use Cases 2/3 (send):
- user can send tweets
- tweets are based on words, each either
  a text "example", tag "#example", user reference
  "@example" or URL http://example.com
- tweets are either public broadcast or
  personal/direct messages
- user can re-tweet a message of others

Use Cases 3/3 (query):
- user can view timeline
  (chronological tweets of others he follows)
- user can search for tweets
  (by keyword "foo", tag "#foo", or user "@foo")
- user can view tag cloud

Frontends/Clients:
- mobile app (iOS, Android)
- desktop app (Windows, Mac OS X)
- web app
- embedded web widget
  (query use cases only)

Current Demand (as of 2012):
- 140M user profiles
- 400M tweets/day
- 6393 tweets/second peak
- 140 characters/tweet
- 30K queries/second
- 300 GB/hour data in total
- 4,4 tweets/day/user on average
- 103,4 follower/user
- < 5s tweet-write-to-read-delay

Future Demand:
- quadratic user and traffic growth

Conversion & Normalization

So what? What are we talking about?

Business-Given Key Figure → Technology-Given Key Figure

Technology-Given Key Figure Catalogue

Resulting Architecture Crux Figures ← Resulting Calculated Figure ← Intermediate Calculated Figure

Calculation (Example)

Twitter Information

6.393	tweets/second peak
400.000.000	tweets/day (write)
2.592.000.000	queries/day (read)
6,5	factor read/writes

140	chars/tweet
4.630	tweets/second (write)
30.000	queries/second (read)
10	tweets/query

4,4	tweets/day/user
2,4	tweets/day (M. Fowler)
0,8	tweets/day (R. Engelschall)
621.880.000	tweets/day (average) total
64,3%	users are active at all
277.778	users/minute active

Traffic Bandwidth

350%	overhead HTTP+TCP+IP+Ethernet
2,2	MB/s (write)
140,2	MB/s (read)
142,4	MB/s

10000	Mbps	1250	MB/s
1000	Mbps	125	MB/s
100	Mbps	12,5	MB/s
10	Mbps	1,25	MB/s

Storage Requirements (static)

140.000.000	user profiles
52.000	chars/users for profile
6,62	TB profile (total)

103,4	follower/user
14.474.600.000	user follow links
32	bytes/link
0,42	TB links (total)

Storage Requirements (dynamic)

300	GB/hour data in total
214	TB/month data in total

200%	overhead storage
1265,9	KB/s tweets
104,3	GB/day tweets
3,1	TB/month tweets

200	chars/log entry
6763,6	KB/s log
557,3	GB/day log
16,6	TB/month log

Computing Hardware Requirements

2000	requests/sec (read) AS performance
100	requests/sec (write) As performance
15,0	servers for writes
46,3	servers for reads

Storage Hardware Requirements

0,3	TB/disk (15K rpm)
8,0	disks/server
2,4	TB/server
8,2	server/month (new)

9,2%	ratio business data
90,8%	ratio infrastructure data

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 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

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

Weighted Decision Matrix

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

$C_{i=1..m}$: Criteria i

$A_{j=1..n}$: Alternative j

$w_{i=1..m}$: in [1/4, 1/2, 1, 2, 4]: Weighting i

$E_{i,j}$ in { -2, -1, 0, +1, +2 }: Evaluation i,j

$R_{j=1..n} = SUM_{i=1..m}(w_i * E_{i,j})$: Rating j

$R_{best} = MAX_{j=1..n}(R_j)$: Best Rating

Light-Weight Alternative: qualitatively cherry-picking major positive/negative backing criterias

	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}

Notice

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

Best Practice Rules

Rule 1: the alternatives have to be really **reasonably comparable**.

Rule 2: the **best** rating should be a least **10% above** the **second** best rating.

Rule 3: the **best** rating should cover at least **80%** of the **requirements**.

Rule 4: the Weighted Decision Matrices should cover at least al**l grand decisions**.

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)

Decision Making Process

Establish Criterias → Organize Criterias Hierarchically → Weight Criterias (simply or AHP)

Find Alternatives → Sort Alternatives Sequentially → Evaluate Alternatives against Criterias → Determine Alternatives Ranking

---

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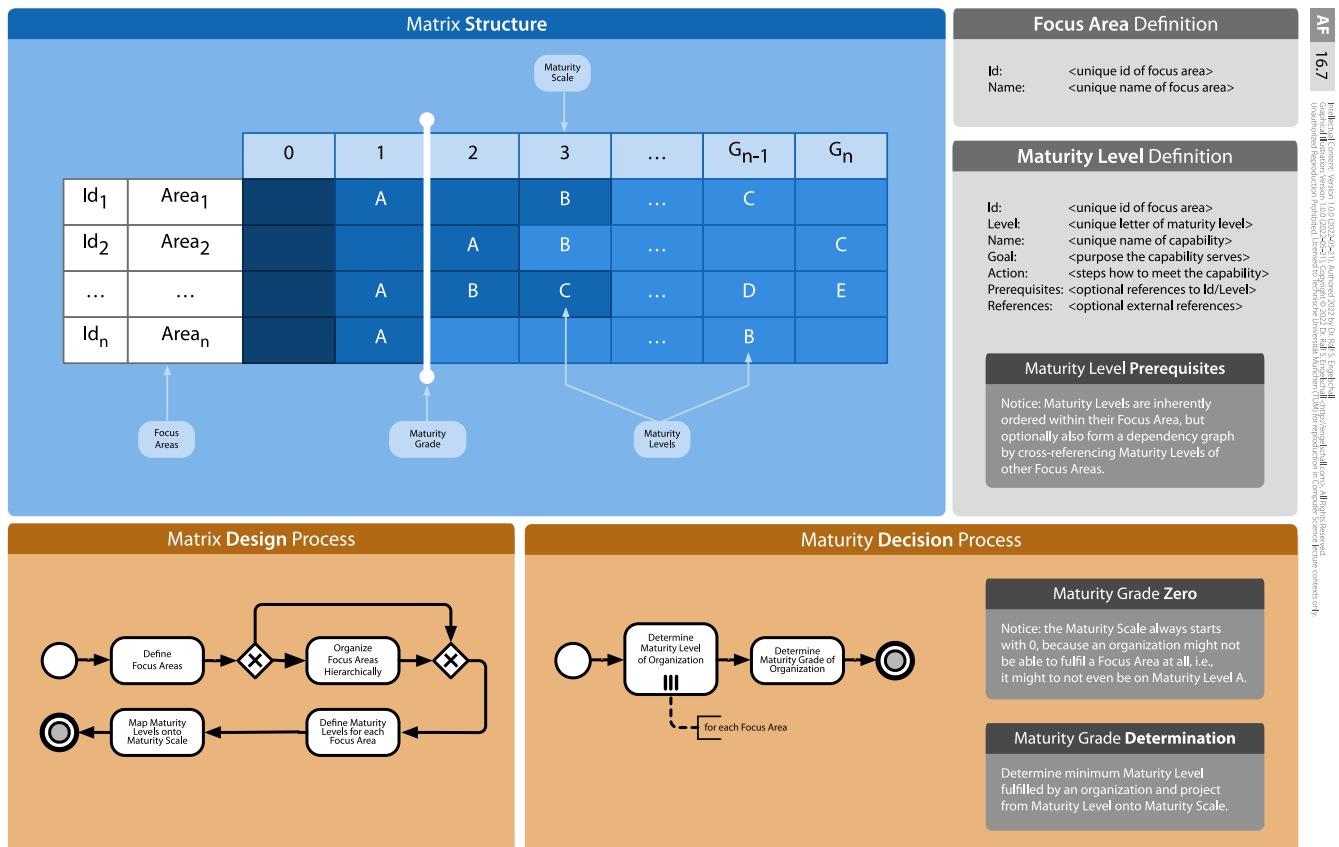
Subsequently, one determines a weighting for each criterion, which indicates how strongly the criterion is considered in the decision-making process.

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

❷ 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

TECHNISCHE UNIVERSITÄT MÜNCHEN

Matrix Structure

Maturity Scale

		0	1	2	3	…	G_{n-1}	G_n
Id_1	$Area_1$		A		B	…		C
Id_2	$Area_2$			A	B	…		C
…	…		A	B	C	…	D	E
Id_n	$Area_n$		A			…	B	

Focus Areas

Maturity Grade

Maturity Levels

Focus Area Definition

Id: <unique id of focus area>
Name: <unique name of focus area>

Maturity Level Definition

Id: <unique id of focus area>
Level: <unique letter of maturity level>
Name: <unique name of capability>
Goal: <purpose the capability serves>
Action: <steps how to meet the capability>
Prerequisites: <optional references to Id/Level>
References: <optional external references>

Maturity Level Prerequisites

Notice: Maturity Levels are inherently ordered within their Focus Area, but optionally also form a dependency graph by cross-referencing Maturity Levels of other Focus Areas.

Matrix Design Process

Define Focus Areas

Organize Focus Areas Hierarchically

Map Maturity Levels onto Maturity Scale

Define Maturity Levels for each Focus Area

Maturity Decision Process

Determine Maturity Level of Organization

Determine Maturity Grade of Organization

for each Focus Area

Maturity Grade Zero

Notice: the Maturity Scale always starts with 0, because an organization might not be able to fulfil a Focus Area at all, i.e., it might to not even be on Maturity Level A.

Maturity Grade Determination

Determine minimum Maturity Level fulfilled by an organization and project from Maturity Level onto Maturity Scale.

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 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 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)?

Vote is a portable mobile-first designed application for easily performing anonymous online votings within a small group of people to figure out their opinions or moods.

Votings are created in advance, executed at a certain time, conducted by the users, and then finally reported.

① Elevator Pitch RE D

Name, Purpose, Motivation, Actors, Devices.

Rationale: Roughly describe the purpose and primary motivation.

Format: Prose Abstract

② Crux Flash RE D

Functionality, Cruxes.

Rationale: Roughly describe the functionality and the cruxes.

Format: Prose Abstract

Votings can be quickly accessed by QR-code or URL and are based on one or more questions and corresponding multiple-choice-based answers.

Votings are interactively conducted, and answers are received and reported either asynchronously in batches (offline voting) or even synchronously in real-time (online voting).

Actor Role	Use–Case
User	Register Account
	Recover Account
	Configure Account
	Login Account
	Logout Account
Author	Create Voting
	Grant Voting Access
Supervisor	Execute Voting
	Enable Question
	Display Result
Voter	Vote Question
	Display Result

③ Customer Journey UX D

Actor Roles, Use-Cases.

Rationale: Sketch the customer journey through major use-cases.

Format: 2xN Table or UML UC Diag.

④ Dialog Storyboard UX D

Dialogs, Interaction, Control Flow.

Rationale: Illustrate the major user interface dialogs (or dialog types).

Format: Wireframe Graph Diagram

Quality	Expectation
Cross-Platform Client	yes
Non-Cleartext Password Storage	yes
Minimum Concurrent Voters (people)	50
Maximum Display Result Latency (sec)	1

⑤ Quality Requirements RE T

Qualities, Expectations.

Rationale: List requirements on the major non-functional qualities.

Format: 2xN Table

⑥ System Architecture SY T

Actors, Systems, Zones, Programs.

Rationale: Illustrate the major system architecture components.
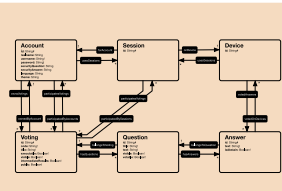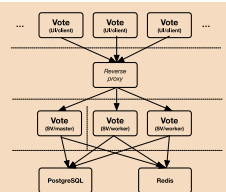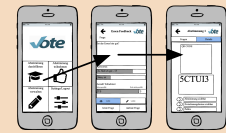
Format: Boxes'n'Lines Diagram

Aspect	Amount	Size	Total Size	Unit
Account Data	10.000	256	2.560.000	B
Voting Data	10.000	1.024	10.240.000	B
Server RAM Usage	200	20	4.000	MB

⑧ Sizing Sketch SW T

Aspects, Amounts, Sizes, Total Sizes, Units.

Rationale: Sketch the sizing of major entities and system parts.

Format: 5xN Table

⑦ Data Model SW T

Entities, Relationships.

Rationale: Model major data entities and their relationships.

Format: UML Class Diagram

(Example)   (Method)   (Example)

① Creation Step   RE Requirements Engineering   UX User Experience   SY Systems Architecture   SW Software Architecture   D Domain Scope   T Technology Scope

In order to understand a Business Information System to be created at a very early stage, a so-called **Big Picture** should be created. This can be developed according to the method **8-D** of Dr. Ralf S. Engelschall, which consists of 8 dimensions (4 domain-specific and 4 technical ones).

In the first step, an **Elevator Pitch** is created, which is defined by **Name**, **Purpose**, **Actors** and **Devices** to give a rough overview of the solution. In the second step, an additional **Crux Flash** is created, which roughly describes the functionality and its main cruxes. Both steps usually consist of only a few prose sentences.

In the third step, a **Customer Journey** is outlined, which is describe, via **Actor Roles** and **Use Cases**, which use cases the different actor roles experience. This usually consists of only a 2-column table.

In the fourth step, a **Dialog Storyboard** is illustrated, which is shown via **Dialogs**, **Interactions** and **Control Flow**, and which illustrates the user interface the application offers. This usually consists of a graph of **Wireframes** (intentionally very rough and fuzzy sketches of the dialogs).

In the fifth step, the **Quality Requirements** are listed via **Qualities** and **Expectations**, to show which expectations exist concerning non-functional quality properties. These are usually executed as a 2-column table.

In the sixth step, the **System Architecture** is illustrated, which via **Actors**, **Systems**, **Tiers/Areas** and **Programs** shows, which main components will exist on the level of System Architecture. This usually consists of a "Boxes'n'Lines" diagram.

In the seventh step, the **Data Model** is modeled, which, via **Entities** and **Relationships**, shows which main domain-specific classes and relationships the data of the application has. This is usually represented as a "UML Class Diagram".

In the eighth and last step, a **Sizing Sketch** is created, which shows via **Aspects**, **Amounts**, **Sizes**, **Total Sizes** and **Units**, which orders of magnitude (in the "worst case") are to be expected with the data and the system components. This usually consists of a 5-column table.

Questions

❓ What are the eight dimensions of the **8-D** model supposed to do?