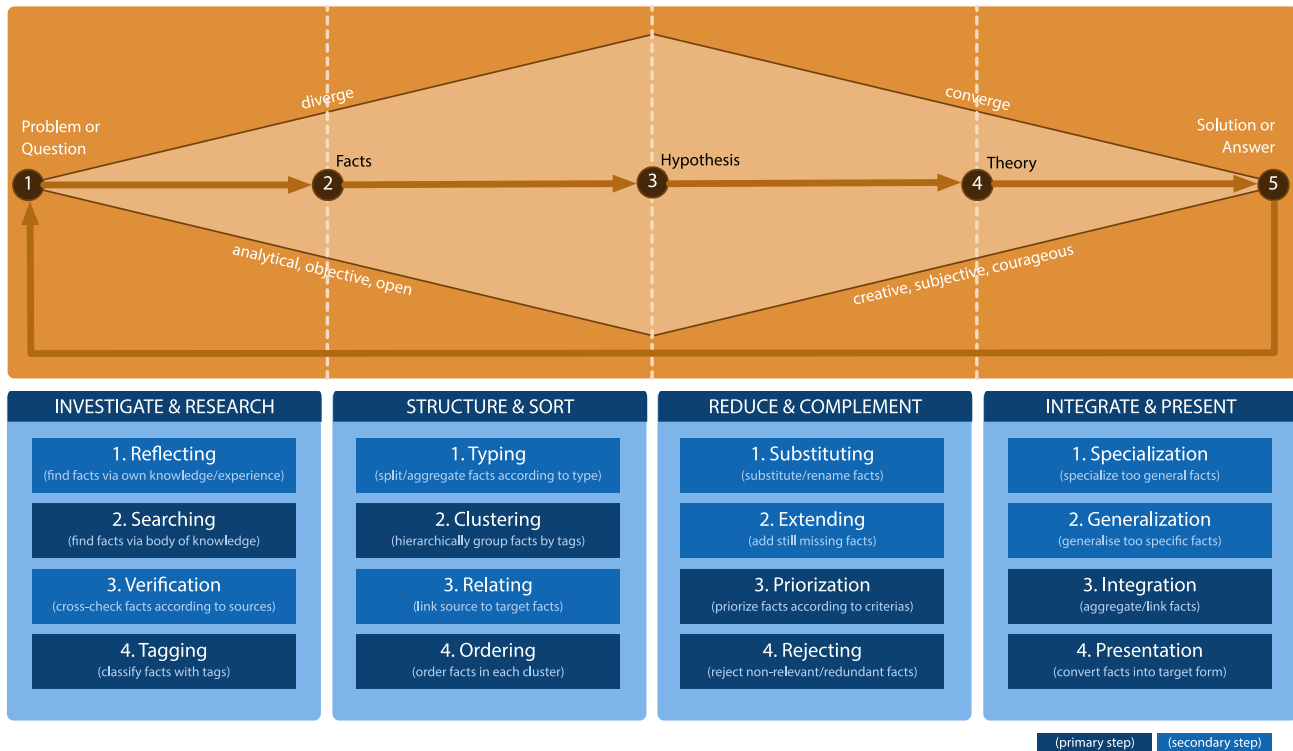




Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall



Der Architekt muß regelmäßig bestimmte Probleme oder Fragestellungen "klardenken". Dazu bietet es sich an, den vierstufigen **Think Clearly** Prozess, einmalig oder bei Bedarf sogar iterativ, zu durchlaufen. Der Prozess besteht aus vier klar unterschiedenen Disziplinen.

In den ersten beiden Disziplinen **Investigate & Research** und **Structure & Sort** versucht man analytisch, objektiv und offen zu agieren und bzgl. des Problems bzw. der Fragestellung zunächst zu **divergieren**, d.h. viele Fakten zu sammeln und daraus eine Hypothese durch Strukturierung und Sortierung aufzustellen.

In den letzten beiden Disziplinen **Reduce & Complement** und **Integrate & Present** versucht man dann kreativ, subjektiv und mutig zu agieren und bzgl. des Problems bzw. der Fragestellung abschließend zu **konvergieren**, d.h. die Hypothese zur in sich stimmigen Theorie zu reduzieren und dann zur Lösung bzw. der Antwort zu integrieren.

In der ersten Disziplin **Investigate & Research** findet man Fakten über eigenes Wissen und eigene Erfahrung (**Reflecting**) oder über eine Recherche in fremden Quellen (**Searching**), verifiziert die Fakten über weitere Quellen (**Verification**) und klassifiziert die Fakten über die Anreicherung mit Tags (**Tagging**).

















In der zweiten Disziplin **Structure & Sort** teilt oder aggregiert man die Fakten für möglichst gute "Sortenreinheit" (**Typing**), gruppiert die Fakten hierarchisch über Tags (**Clustering**), verbindet verwandte Fakten gruppenübergreifend (**Relating**) und sortiert die Fakten in jeder Gruppe.

In der dritten Disziplin **Reduce & Complement** ersetzt man bei Bedarf Fakten oder benennt Fakten um (**Substituting**), fügt noch fehlende Fakten hinzu (**Extending**), priorisiert die Fakten anhand von bestimmten Kriterien (**Priorization**) und verwirft bei Bedarf nicht relevante oder redundante Fakten (**Rejecting**).

In der vierten und letzten Disziplin **Integrate & Present** spezialisiert man bei Bedarf zu allgemeine Fakten (**Specialization**), verallgemeinert man bei Bedarf zu spezielle Fakten (**Generalization**), aggregiert oder verbindet man bei Bedarf Fakten (**Integration**) und konvertiert Fakten in ihre Zieldarstellung (**Presentation**).

Fragen

- ?** Im **Think Clearly** Prozess zum "Klardenken" soll man über die ersten beiden und die letzten beiden Disziplinen **wie denken**?

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. 	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. 	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.

Bei den **Problem Solving Heuristics** handelt es sich um Erfahrungs-basierte Techniken oder Strategien, die man zur Problemlösung einsetzen kann, wenn man mit anderen Ansätze nicht weiterkommt.

Die Heuristiken dienen vor allem der Inspiration, um sich bei einem Problem nicht zu lange im Kreis zu drehen und stattdessen einen neuen Ansatzpunkt zur Problemlösung zu finden ("If you find yourself in a hole, stop digging!").

Bei **Research** recherchiert man Fakten, bei **Brainstorming** schlägt man eine große Anzahl von spontanen Lösungsideen vor, bei **Analogy** denkt man an ähnliche bereits gelöste Probleme und bei **Reduction** transformiert man das Problem in ein bereits gelöstes Problem.

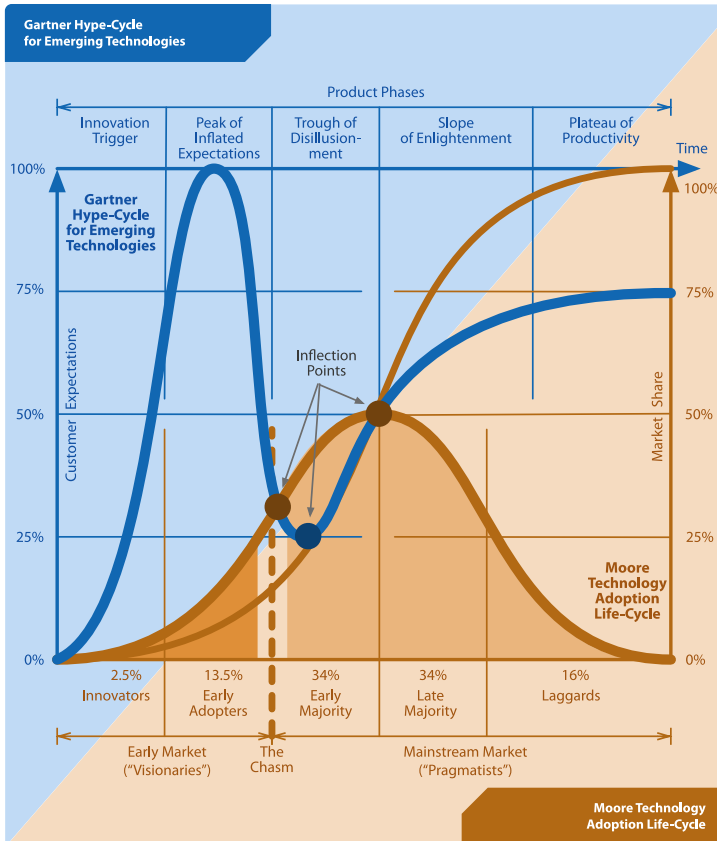
Bei **Abstraction** löst man das Problem zuerst in einem abstrakteren Modell, bei **Generalization** verallgemeinert man das Problem zu einem mit weniger Spezialfällen, bei **Specialization** versucht man über einen Spezialfall inspiriert zu werden und bei **Variation** versucht man die eigene Perspektive auf das Problem zu verändern.

Bei **Lateral Thinking** geht man das Problem bewusst indirekt und kreativ an, bei **Hypothesis Proof** sucht man die Lösung über einen Beweis für oder gegen eine mögliche fiktive Lösung, bei **Root Cause** geht man dem Problem schrittweise auf den Grund und bei **Means End** versucht man sich der Lösung in kleinen Schritten zu nähern.

Bei **Backward Search** versucht man von einer fiktiven Lösung aus rückwärts auf den Weg dorthin zu kommen, bei **Backtracking** wählt man bei einem Fehlschlag in Richtung der Lösung einen nur teilweise neuen Weg, bei **Divide & Conquer** zerlegt man das große Problem in kleinere und leichter zu lösende Teilprobleme und bei **Trail & Error** versucht man, als letzte aller Möglichkeiten und/oder wenn der Lösungsraum klein genug ist, alle Lösungskombinationen einfach durchzuprobieren.

Fragen

❓ Wann ist die recht profane **Problem Solving Heuristic** namens **Trial & Error** akzeptabel?



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://gtm.it/366BTAK>
[2] <https://bit.ly/2q24Lke>
[3] <https://bit.ly/2N3JB1t>
[4] <https://bit.ly/2NURNT7>
[5] <https://bit.ly/24IMKEW>

Man kann eine Technologie (bzw. ein konkretes technologisches Produkt) sehr gut über zwei Modelle, die **Technology Life-Cycles**, klassifizieren: den **Gartner Hype-Cycle for Emerging Technologies** und den **Moore Technology Adoption Life-Cycle**.

Der **Gartner Hype-Cycle for Emerging Technologies** zeigt den üblichen Lebenszyklus einer Technologie über die zeitlichen **Product-Phases** (in der x-Achse) **Innovation Trigger**, **Peak of Inflates Expectations**, **Trough of Disillusionment**, **Slope of Enlightenment** und **Plateau of Productivity**, und über die **Customer Expectations** (in der y-Achse). Er bildet damit primär den Reifegrad einer Technologie ab und zeigt, welche Erwartungen die Technologie am Markt zum jeweiligen Zeitpunkt hervorruft.

Der **Moore Technology Adoption Life-Cycle** zeigt die Akzeptanz der Technologie in verschiedenen Arten von Märkten. Diese Märkte sind durch die grundlegend verschiedenen agierenden Marktteilnehmer **Innovators**, **Early Adopters**, **Early Majority**, **Late Majority** und **Laggards** definiert, wobei üblicherweise eine gewisse schwer zu überbrückende Kluft (**The Chasm**) zwischen dem **Early Market** aus Visionären und dem **Mainstream Market** aus Pragmatikern existiert. Um diese Kluft zu meistern, muß eine Technologie meist erst in einer zweiten Generation entwickelt werden.

Der **Moore Technology Adoption Life-Cycle** hat auch einen Bezug zum **Innovators Dilemma**. Denn man kann den maximal erzielbaren Marktanteil einer Technologie über die Zeit in Form einer S-Kurve verstehen. Die Knackpunkte für eine Technologie sind hierbei bei etwa 25% (**The Chasm**) und 50% Marktanteil. Außerdem zeigt diese S-Kurve das **Innovators Dilemma**, also die Tatsache, daß eine neue Technologie immer eine zeitliche Durststrecke in Nischen des **Early Market** überbrücken muss, bevor sie größere Marktanteile auf dem **Mainstream Market** erzielen kann.

Fragen

❓ Welches Modell eines **Technology Life-Cycles** bildet den Reifegrad einer Technologie über die Zeit ab?

❓ Welches Modell eines **Technology Life-Cycles** bildet die Akzeptanz einer Technologie in verschiedenen Märkten ab?

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

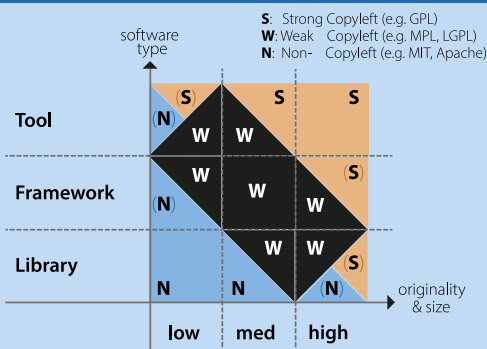
Pragmatism
Business
Engineering

Industry
Private
Science

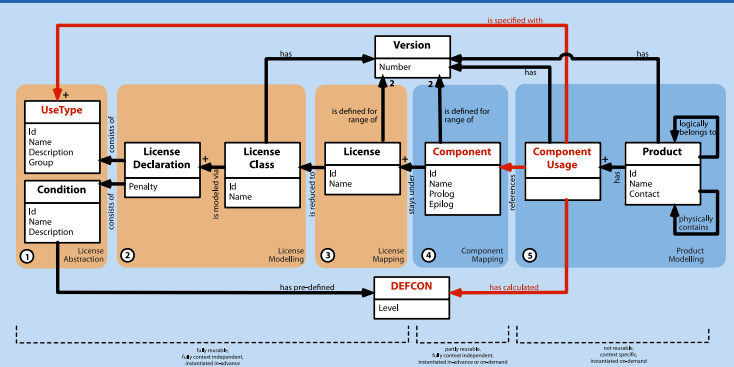
Most Popular Open Source Licenses



Choosing an Open Source License



License Compliance Checking Meta-Model



Bei **Open Source Software** handelt es sich eine Software, die unter eine **Open Source License** gestellt wurde. Alle als **Open Source License** anerkannten Lizenzen genügen der **Open Source Definition**, welche u.a. besagt, daß die Software frei im Quellcode (weiter-)verteilt und verändert werden darf und es vor allem keine Diskriminierung von Personen, Gruppen oder Einsatzzwecken geben darf.

In der Praxis kennt man drei **Open Source Personality Streams**: **Software Sharing** mit dogmatischen und politischen Personen, welche für die soziale Gerechtigkeit kämpfen; **Software Hacking** mit fundamentalen und künstlerischen Personen, welche Software mit maximalem Anspruch entwickeln; und **Software Engineering** mit pragmatischen Personen, welche die Software in der Praxis einsetzen.

Es gibt hunderte **Open Source Licenses**. Allerdings kann man diese in ganz wenige Klassen einteilen und nach ihrer Strenge sortieren, d.h. wie stark sie die Software selbst schützen. Dabei unterscheidet man generell zwischen Lizenzen ohne und mit einem sog. **Copyleft**-Effekt. Dieser besteht aus Lizenzklauseln, um die ursprüngliche Software frei zu behalten (im Sinne von Freiheit und Verfügbarkeit, nicht im Sinne von kostenlos) und zusätzlich alle Modifikationen und Erweiterungen an der Software ebenfalls frei zu behalten.

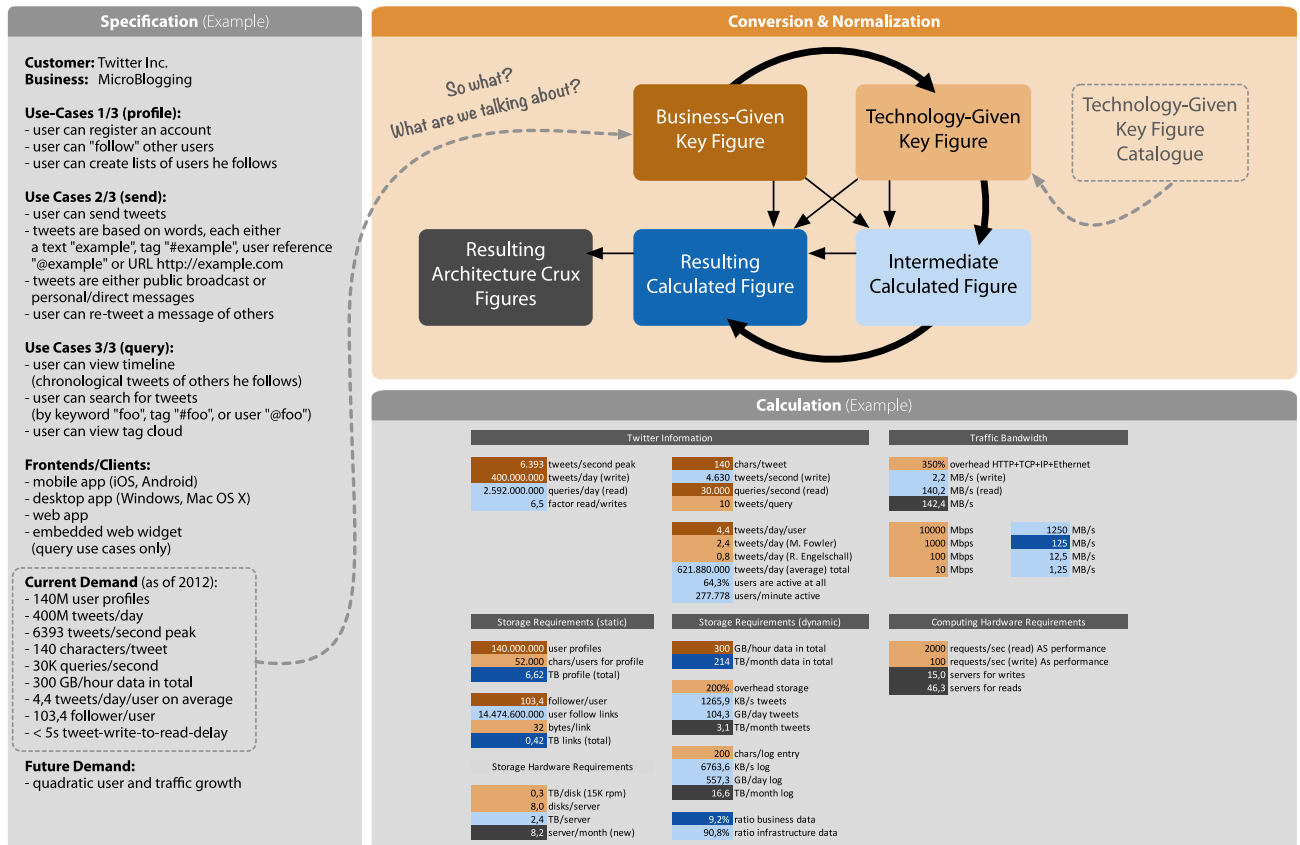
Die schwächste Lizenz in der Praxis ist **Creative Commons Zero (CC0)** (bzw. **Public Domain**), die jedermann effektiv alles erlaubt.

Die stärkste Lizenz ist die **Affero General Public License (AGPL)**, die eine Software sogar im Falle der Nutzung in Form von Software as a Service (SaaS) schützt. An der Copyleft-Grenze liegt die **Apache License**, welche noch keinen Copyleft-Effekt besitzt, aber die Software und den Urheber versucht noch maximal zu schützen.

In der Praxis unterscheidet man Lizenzen mit keinem, schwachem oder starkem Copyleft. Um für eine Software zu entscheiden, unter welcher Klasse von Lizenz man sie veröffentlicht, unterscheidet man zwei Dimensionen: einerseits die Art der Software (**Tool**, **Framework** oder **Library**) und andererseits die Schöpfungshöhe der. Ein **Tool** oder ein **Framework** mit mittlerer oder hoher Schöpfungshöhe stellt man üblicherweise unter schwaches oder sogar starkes Copyleft, um die Software und den Urheber maximal zu schützen. Eine **Library** oder ein **Framework** mit mittlerer oder niedriger Schöpfungshöhe stellt man dagegen eher unter schwaches oder sogar kein Copyleft, um einen maximalen Verbreitungsgrad der Software zu erreichen.

Fragen

- Wie nennt man den Effekt in Lizenzen von **Open Source Software**, bei dem dafür gesorgt wird, daß die Software frei bleibt (im Sinne von Freiheit und Verfügbarkeit, nicht im Sinne von kostenlos) und zusätzlich alle Modifikationen und Erweiterungen ebenfalls frei bleiben?



Um ein besseres "Gefühl" für den Umfang und den Schwierigkeitsgrad einer zu entwickelnden Architektur zu bekommen, bietet es sich an, eine **Back of the Envelope Calculation** ("Überschlagsrechnung") zu machen. Hierzu wird methodisch wie folgt vorgegangen: in einem Spreadsheet wird eine zweispaltige Tabelle angelegt, bei der die erste Spalte die Zahl und die zweite Spalte die Einheit aufnimmt.

Nun werden im ersten Schritt die **Business-Given Key Figures**, also die fachlich vorgegebenen bzw. bekannten Zahlen in die Tabelle als erste Zeilen eingetragen. Sie bekommen die erste Farbe zur Unterscheidung.

Weil einem diese Zahlen üblicherweise noch zu wenig sagen, werden in einem zweiten Schritt verschiedene **Technology-Given Key Figures** als Zeilen eingetragen. Diese werden eventuell aus bestehenden Katalogen entnommen oder stehen über die eigene Erfahrung zur Verfügung. Sie bekommen die zweite Farbe zur Unterscheidung und dienen vor allem als Vergleichsgrößen zu den **Business-Given Key Figures**.

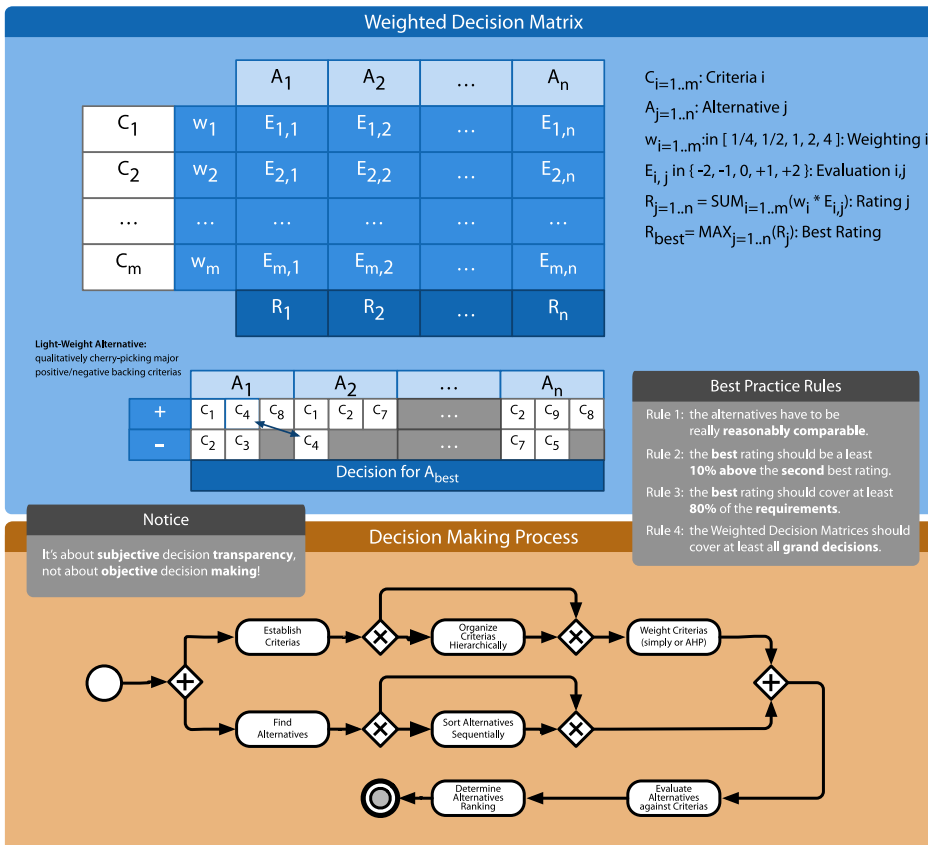
Im dritten Schritt wird man nun üblicherweise sowohl die **Business-Given Key Figures** als auch die **Technology-Given Key Figures** einheitengetreu umrechnen und miteinander vergleichen. Die Zwischenergebnisse, genannt **Intermediate Calculated Figures**, sind Spreadsheet-Zellen mit Formeln, welche die dritte Farbe zur Unterscheidung bekommen.

Immer wenn eine **Intermediate Calculated Figure** (ggf. auch schon eine **Business-Given Key Figure** oder eine **Technology-Given Key Figure**) einen entscheidenden Hinweis oder eine Erkenntnis liefert, wechselt man die Zeile auf die vierte Farbe. Sollte diese Erkenntnis potenziell eine wesentliche Relevanz für die spätere Architektur haben (und somit einen Knackpunkt darstellen), wechselt man die Zeile auf die fünfte Farbe, welche die **Resulting Architecture Crux Figure** zeigt.

Anschließend kann man optional die verschiedenen Zeilen zu logischen Gruppen im Spreadsheet bündeln, damit das Spreadsheet übersichtlicher wird.

Fragen

- ❓ Welche Methode kann man anwenden, um ein besseres "Gefühl" für den Umfang und den Schwierigkeitsgrad einer zu entwickelnden Architektur zu bekommen?



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)

Um qualitative Entscheidungen transparent und nachvollziehbar (aber nicht zwingendermaßen objektiv) zu treffen und gleichzeitig die Entscheidungsfindung zu dokumentieren, kann man die Methode der **Weighted Decision Matrix** (gewichtete Entscheidungsmatrix) anwenden.

Voraussetzung ist, daß die zu treffende Entscheidung die Auswahl einer von vielen Alternativen ist. Diese trägt man in ein Spreadsheet als Spalten ein. Optional können die Alternativen in eine sinnvolle Reihenfolge gebracht werden.

Dann legt man verschiedene Kriterien fest, welche für die Entscheidung herangezogen werden sollen. Ziel ist es, mit möglichst wenig Kriterien, die Alternativen möglichst gut zu unterscheiden. Jedes Kriterium bekommt ein Gewicht. Optional können die Kriterien hierarchisch gruppiert und die Gruppen in eine sinnvolle Reihenfolge gebracht werden.

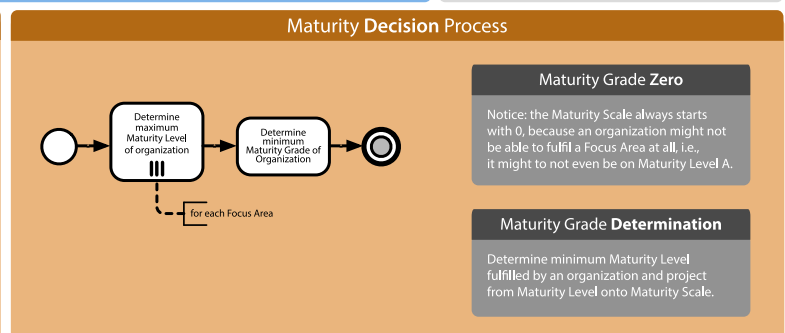
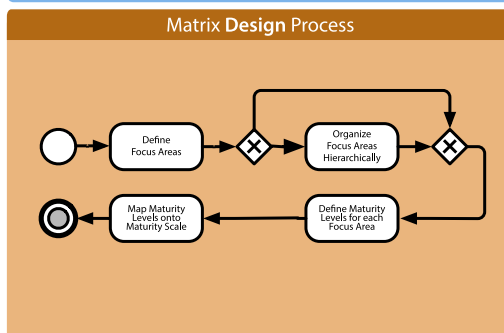
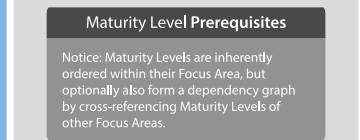
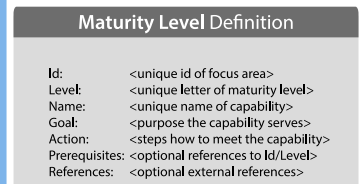
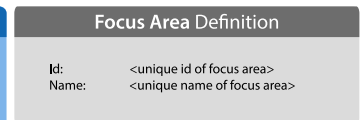
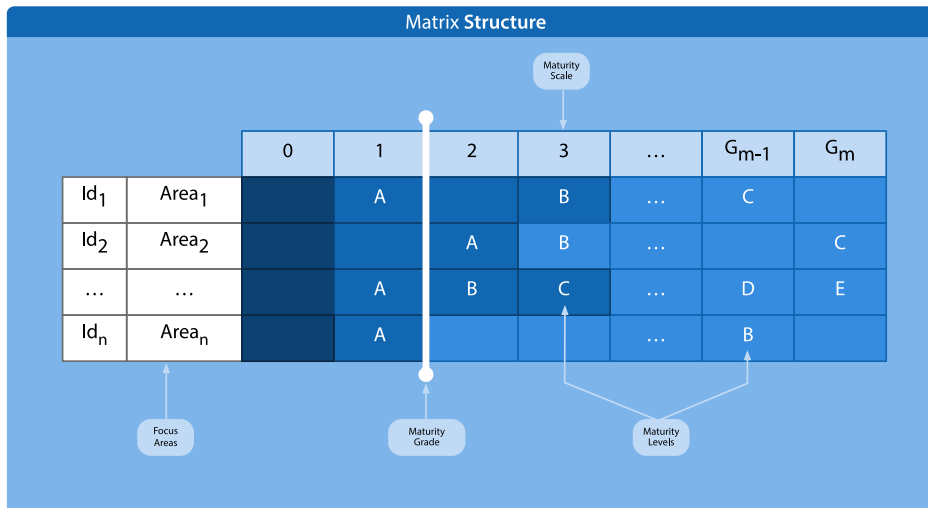
Anschließend legt man für jedes Kriterium ein Gewicht fest, welches angibt, wie stark das Kriterium in die Entscheidungsfindung einfließt.

Nun werden alle Alternativen gegenüber allen Kriterien bewertet. Bei wenigen Kriterien und vielen Alternativen kann man pro Alternative alle Kriterien bewerten. Bei vielen Kriterien bietet es sich an, pro Kriterium alle Alternativen zu bewerten. Als Bewertungsskala bietet sich -2, -1, 0, +1, +2 an, um eine Mitte (0), positive/negative Bewertungen (+1, -1) und positive/negative Superlative (+2, -2) zu haben.

Zum Schluß wird für jede Alternative die Produktsumme aus der Kriterium-Gewichts-Spalte und der Alternativen-Bewertungs-Spalte berechnet. Die Entscheidung fällt dann für die Alternative mit dem Maximalbetrag in der Produktsumme.

Fragen

- Wie kann man die qualitative Entscheidung für die Auswahl einer aus vielen Alternativen transparent und nachvollziehbar treffen und gleichzeitig dokumentieren?



Das Focus Area Maturity Model (FAMM) ist eine Methode, um den Reifegrad einer Organisation bezüglich eines bestimmten Themenbereichs zu bestimmen.

Die Struktur eines FAMM ist eine Matrix aus horizontalen Focus Areas und ihrer Maturity Levels und möglichen vertikalen Maturity Grades auf einer Maturity Scale. Pro Focus Area kann es einen oder eine beliebige Anzahl an Maturity Levels geben und deren Positionierungen auf der Maturity Scale orientierten sich an der Wichtigkeit der Focus Areas und dem Bezug zwischen den Focus Areas und ihren Maturity Levels. Diese Matrix wird in einem ersten Schritt für einen Themenbereich entworfen und ist danach fest.

Für die Bestimmung des Reifegrads einer Organisation wird pro Focus Area überprüft, welchen maximalen Maturity Level die Organisation erfüllt. Der Maturity Grade der Organisation ergibt sich dann aus dem minimalen Maturity Level über alle Focus Areas und der Projektion dieses Maturity Level auf die Maturity Scale.

Da die Maturity Levels immer überhalb dem Maturity Grade 0 in der Matrix positioniert werden sollten, hat eine Organisation im schlimmsten Fall den Maturity Grade 0, falls sie eine Focus Area gar nicht erfüllt.

Fragen

- ? Von wem kann man mit dem Das Focus Area Maturity Model (FAMM) den Reifegrad bestimmen?

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.

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

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

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

(Example)

1 Elevator Pitch (RE D)

Name, Purpose, Motivation, Actors, Devices.

Rationale: Roughly describe the purpose and primary motivation.

Format: Prose Abstract

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.

5 Quality Requirements (RE T)

Qualities, Expectations.

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

Format: 2xN Table

8 Sizing Sketch (SW T)

Aspects, Amounts, Sizes, Total Sizes, Units.

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

Format: 5xN Table

(Method)

2 Crux Flash (RE D)

Functionality, Cruxes.

Rationale: Roughly describe the functionality and the cruxes.

Format: Prose Abstract

4 Dialog Storyboard (UX D)

Dialogs, Interaction, Control Flow.

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

Format: Wireframe Graph Diagram

6 System Architecture (SY T)

Actors, Systems, Zones, Programs.

Rationale: Illustrate the major system architecture components.

Format: Boxes'n'Lines Diagram

7 Data Model (SW T)

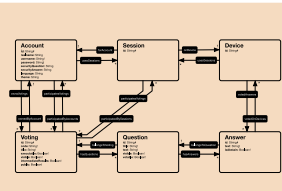
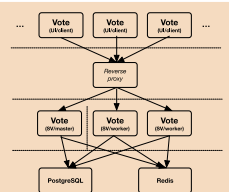
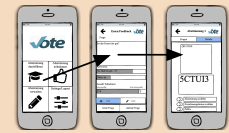
Entities, Relationships.

Rationale: Model major data entities and their relationships.

Format: UML Class Diagram

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).



(Example)

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

Um sehr früh ein zu erstellendes betriebliches Informationssystem zu verstehen, sollte ein sogenanntes **Big Picture** erstellt werden. Dieses kann man nach der Methode **8-D** von Dr. Ralf S. Engelschall entwickeln, welches aus 8 Dimensionen (4 fachlichen und 4 technischen) besteht.

Im ersten Schritt wird ein **Elevator Pitch** erstellt, der über **Name, Purpose, Actors** und **Devices** einen groben Überblick über die Lösung liefert. Im zweiten Schritt wird zusätzlich ein **Crux Flash** erstellt, der über **Functionality** und **Cruxes** grob die Funktionalität und deren wesentliche Knackpunkte aufzeigt. Beide Schritte bestehen üblicherweise aus nur sehr wenigen Prosa-Sätzen.

Im dritten Schritt wird eine **Customer Journey** skizziert, die über **Actor Roles** und **Use Cases** beschreibt, welche Anwendungsfälle die verschiedenen Rollen von Akteuren erleben. Diese besteht üblicherweise nur aus einer 2-spaltigen Tabelle.

Im vierten Schritt wird ein **Dialog Storyboard** illustriert, welches über **Dialogs, Interactions** und **Control Flow** aufzeigt, welche Benutzeroberfläche der Anwendung bietet. Dieses besteht üblicherweise aus einem Graphen von **Wireframes** (absichtlich sehr groben und unscharfen Skizzen der Dialoge).

Im fünften Schritt werden die **Quality Requirements** aufgelistet, welche über **Qualities** und **Expectations** zeigen, welche Erwartungshaltung bezüglich nicht-funktionaler Qualitätseigenschaften existieren. Diese werden üblicherweise als 2-spaltige Tabelle ausgeführt.

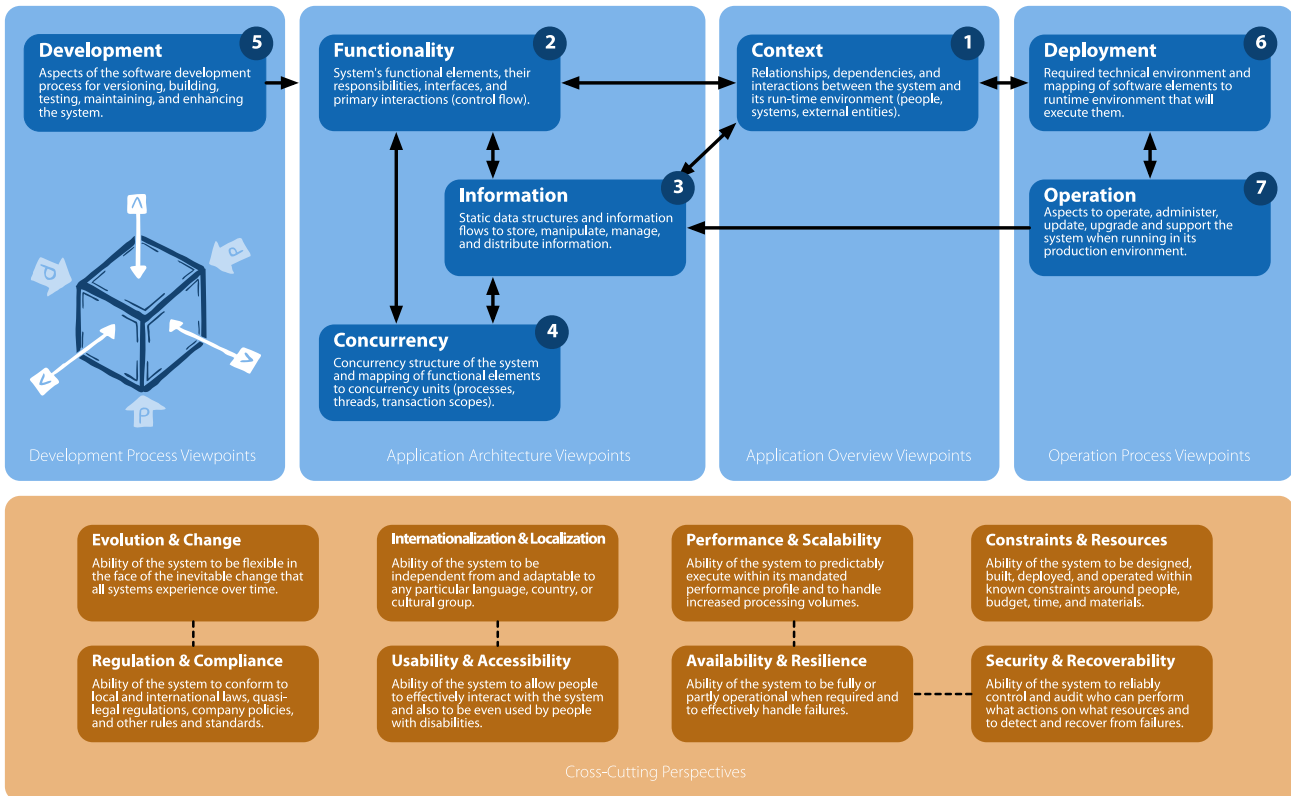
Im sechsten Schritt wird die **System Architecture** illustriert, welche über **Actors, Systems, Tiers/Areas** und **Programs** zeigt, welche hauptsächlichen Komponenten auf der Ebene von System Architecture existieren werden. Diese besteht üblicherweise aus einem "Boxes'n'Lines"-Diagramm.

Im siebten Schritt wird das **Data Model** modelliert, welches über **Entities** und **Relationships** zeigt, welche hauptsächlichen fachlichen Klassen und Beziehungen die Daten der Anwendung besitzen. Dieses wird üblicherweise als "UML Class Diagram" repräsentiert.

Im achten und letzten Schritt wird ein **Sizing Sketch** erstellt, welcher über **Aspects, Amounts, Sizes, Total Sizes** und **Units** aufzeigt, welche Größenordnungen (im "worst case") bei den Daten und den Systemkomponenten zu erwarten sind. Dieser besteht üblicherweise aus einer 5-spaltigen Tabelle.

Fragen

? Was sollen die acht Dimensionen des 8-D Modells leisten?



Um eine **Architecture Description** (im deutschen Kontext üblicherweise **IT-Konzept** genannt) für eine Anwendung zu erstellen, dokumentiert man methodisch über **Viewpoints** (Sichten) und **Perspectives** (Perspektiven).

Erstere sind 2-dimensionale Diagramme mit Erklärung eines einzigen ganz bestimmten Sachverhalts. Letztere sind Erklärungen eines übergreifenden Sachverhalts, welche alle von **Nicht-Funktionalen Anforderungen** abstammen.

Wie viele **Viewpoints** und **Perspectives** tatsächlich dokumentiert werden müssen, hängt vom **Concern** der **Stakeholder** ab! Ein Standard-Satz an **Viewpoints** und **Perspectives** ist der folgende.

Bei den **Viewpoints** bietet es sich an, z.B. 7 konkrete **Views** zu dokumentieren: **Context, Functionality, Information, Concurrency, Development, Deployment** und **Operation**. Die beiden Viewpoints **Functionality** und **Information** sind die beiden wichtigsten, um die Architektur einer Anwendung zu dokumentieren. Sie sollten immer dokumentiert und somit niemals weggelassen werden!

Bei den **Perspectives** bietet es sich an, z.B. 8x2 konkrete **Aspects** zu dokumentieren, welche auf üblichen **Non-Functional Requirements** basieren: **Evolution & Change, Regulation & Compliance, Internationalization & Localization, Usability & Accessibility, Performance & Scalability, Availability & Resilience, Constraints & Resources** und **Security & Recoverability**. Man muss bei den **Perspectives** in der Praxis aber generell alle im Projekt relevanten **Non-Functional Requirements** berücksichtigen und darf sich nicht nur auf diese Üblichen beschränken.

Fragen

- ❓ Welche zwei **Viewpoints** in der **Architecture Description** einer Anwendung sollten immer dokumentiert werden?