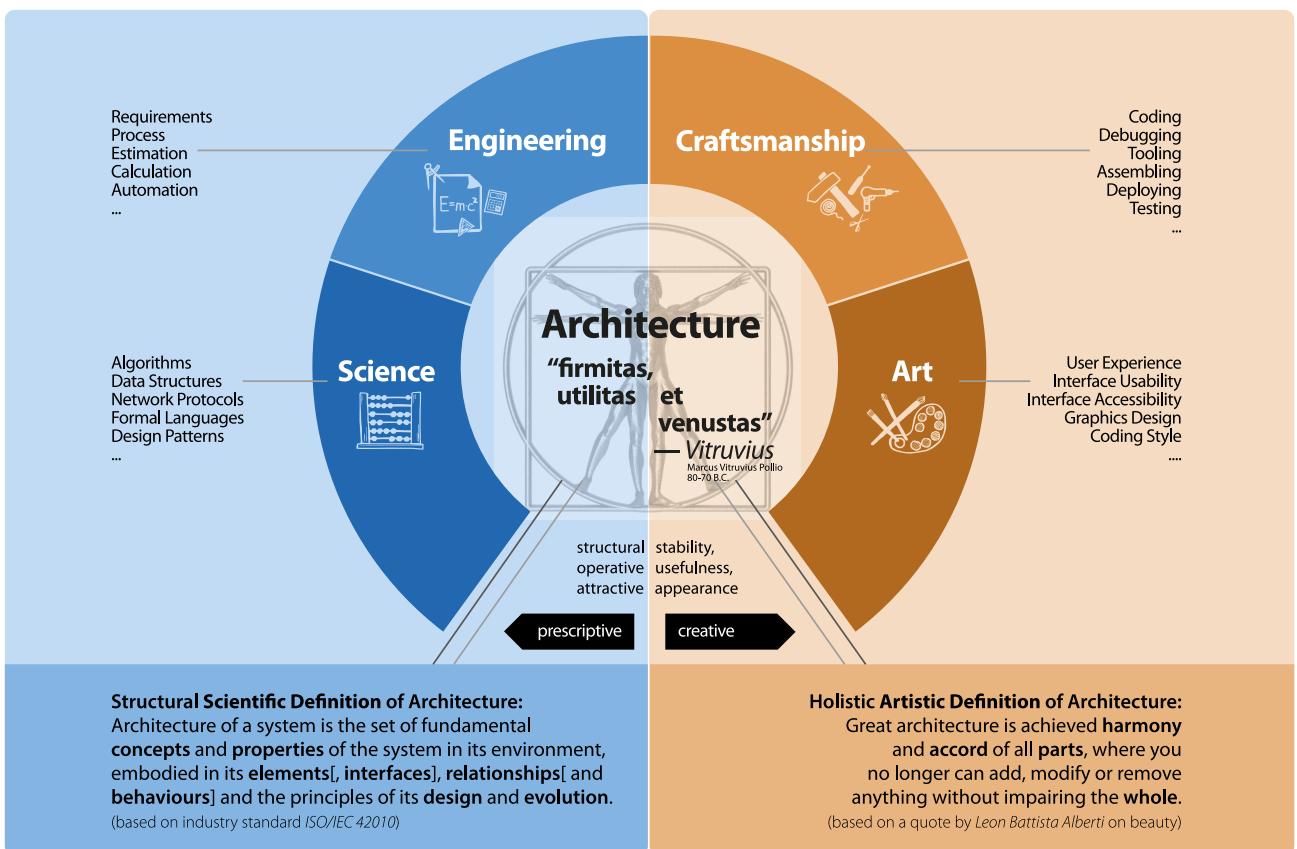




# Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall

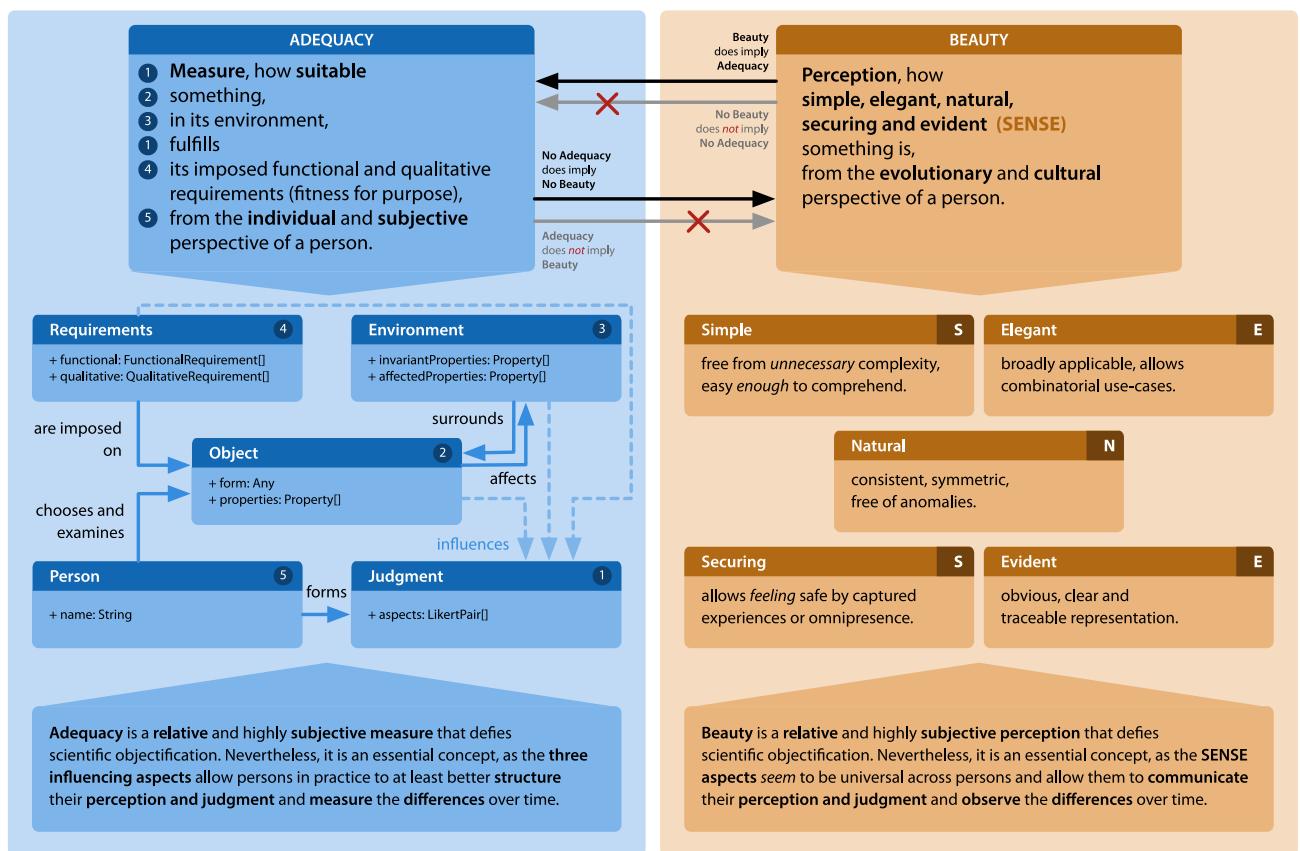


**Architektur** ist nicht einfach zu definieren. Man kann Architektur sowohl **strukturell wissenschaftlich** über messbare Elemente (elements), Schnittstellen (interfaces) und Beziehungen (relationships), als auch **ganzheitlich künstlerisch** über "die Harmonie und den Einklang aller Teile" definieren. Die "Wahrheit" liegt in der Praxis irgendwo dazwischen, denn die beiden Extrema spannen einen breiten Raum auf, in dem alle Lösungen in der Praxis liegen.

Auf der strukturell wissenschaftlichen Seite definiert sich Architektur über die Aspekte **Science** (insbesondere Computer Science) und **Engineering** (insbesondere Software Engineering). Auf der ganzheitlich künstlerischen Seite definiert sich Architektur über die Aspekte **Craftsmanship** (Handwerk, insbesondere Programmieren) und **Art** (Kunst, insbesondere User Experience).

## Fragen

- ?
- Wie kann man **Architektur** definieren?
- ?
- Über welche vier **Aspekte** kann man **Architektur** aufspannen?



Angemessenheit ist definiert als das Maß, wie geeignet etwas, in seiner Umgebung, die gestellten funktionalen und qualitativen Anforderungen erfüllt (Eignung für den Einsatzzweck), aus der individuellen Perspektive einer Person.

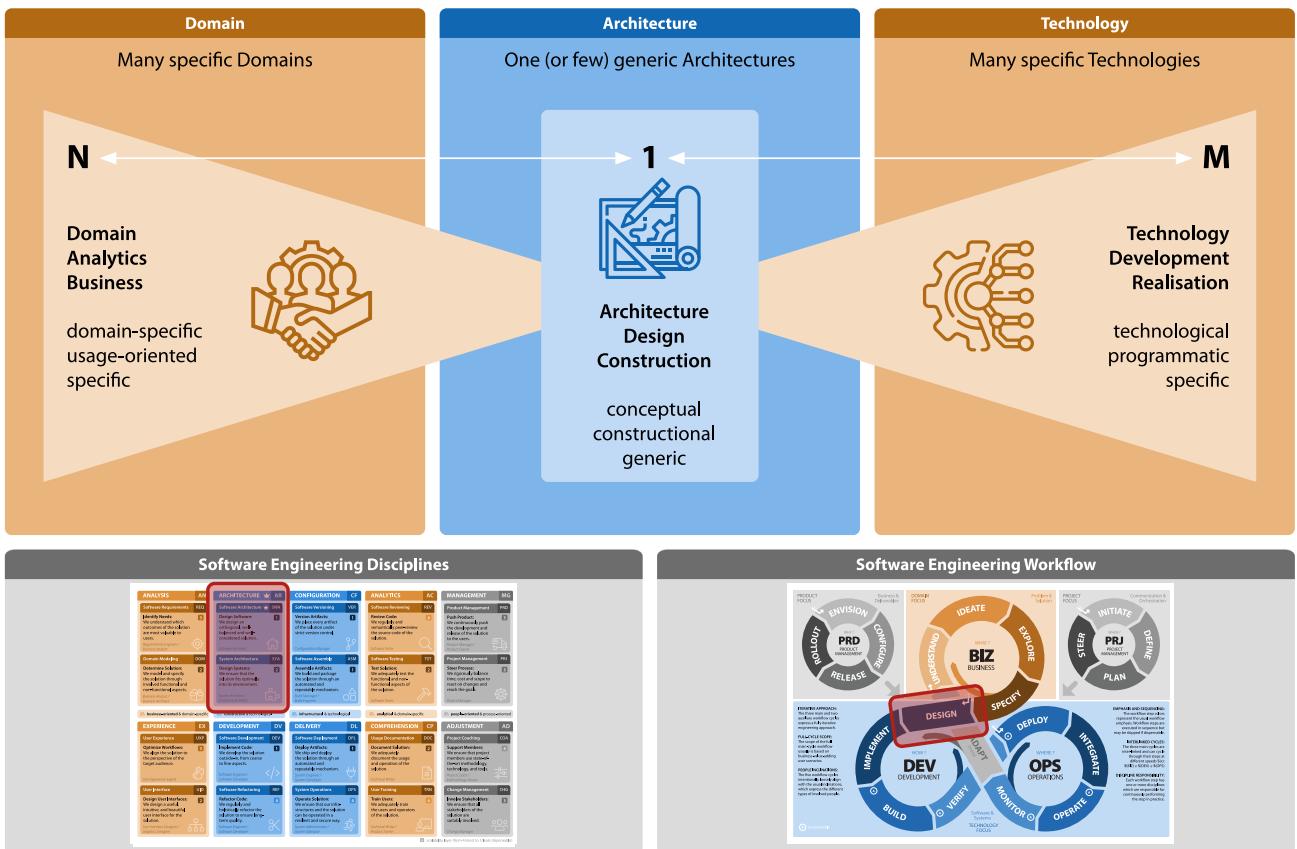
Angemessenheit ist ein relatives und höchst subjektives Maß, das sich einer wissenschaftlichen Objektivierung entzieht. Dennoch ist Angemessenheit ein wichtiges Konzept, da die drei beeinflussenden Aspekte (Anforderungen, Umgebung, Objekt) es den Personen in der Praxis ermöglichen, ihre Wahrnehmung und ihr Urteil zumindest besser zu strukturieren und die Unterschiede über die Zeit zu messen.

Schönheit ist definiert als die Wahrnehmung, wie einfach, elegant, natürlich, sicher und offensichtlich (SENSE) etwas ist, aus der evolutionären Perspektive eines Menschen.

Schönheit ist eine absolute und höchst subjektive Wahrnehmung, die sich einer wissenschaftlichen Objektivierung entzieht. Dennoch ist Schönheit ein wichtiges Konzept, da die SENSE-Aspekte universell über Personen hinweg zu sein scheinen und es ihnen erlauben ihre Wahrnehmung und ihr Urteil mitzuteilen und die Unterschiede über die Zeit zu messen.

## Fragen

- ❓ Kann man Angemessenheit oder Schönheit allgemein messen?
- ❓ Kann man Angemessenheit oder Schönheit im Kontext einer einzelnen Person messen?



(Software- und System-)Architecture gilt als die „Königsdisziplin“ im Software Engineering, da es das zentrale, allgemeine und konzeptuelle Bindeglied zwischen den vielen, potentiellen, spezifischen, realisierten **Fachlichkeiten** und den vielen, potentiellen, spezifischen, realisierenden **Technologien** ist. Die architekturelle Konstruktion einer Anwendung findet im logischen Schritt „Design“ innerhalb des BizDevOps-Workflows im Software Engineering statt.

## Fragen

❓ Wieso gilt Architektur als die „Königsdisziplin“ im Software Engineering?

# Manifesto for IT Architecture

## Continuously Raising the Bar

**Mission** As IT Architects we guide the design, implementation and evolution of IT solutions.

**Entitlement** We continuously strive to raise the bar of professional IT architecture by practicing it and helping others to learn our craft.  
We achieve maximum value for our clients through our work.

**Values** Through this work we have come to value aspects of our craft.  
While we acknowledge the beneficial values in the items on the right, we appreciate the stronger values in the items on the left even more.

|                                   |  |
|-----------------------------------|--|
| <b>Sustainable Concepts</b>       | over <b>Latest Technologies</b>        |
| <b>Pragmatic Making</b>           | over <b>Theoretical Consideration</b>  |
| <b>Constructive Craftsmanship</b> | over <b>Analytical Engineering</b>     |
| <b>Accredited Creativity</b>      | over <b>Achieved Industrialization</b> |
| <b>Proactive Improvement</b>      | over <b>Reactive Correction</b>        |
| <b>Inherent Quality</b>           | over <b>Tested Robustness</b>          |
| <b>Operational Delight</b>        | over <b>Useful Functionality</b>       |



Das **Manifesto for IT Architecture** ist eine Grundsatzklärung für IT-Architektur. Sie besagt zuallererst, daß kontinuierlich die Meßlatte zu erhöhen ist ("Continuously Raising the Bar"), da nach gerade Mal 50 Jahren Software Engineering und Software Architecture wir zwar bereits etliche Best Practices kennen, aber die Disziplin sich sicherlich noch sehr lange weiterentwickeln muss.

Der Auftrag (Mission) an IT-Architekten ist, die Konstruktion, die Implementierung und die Weiterentwicklung/Wartung von IT-Lösungen zu leiten. Der Anspruch (Entitlement) ist es, dabei die Messlatte kontinuierlich zu erhöhen und anderen zu helfen, das "Handwerk" zu erlernen. Selbstredend ist die Tatsache, daß durch die Arbeit von Architekten maximaler Mehrwert für die Kunden erzielt wird.

Die Basis-Werte (Values), welche in diesem Handwerk eine zentrale Rolle spielen und sehr geschätzt werden, sind: **Latest Technologies**, **Theoretical Consideration**, **Analytical Engineering**, **Archived Industrialization**, **Reactive Correction**, **Tested Robustness** und **Useful Functionality**.

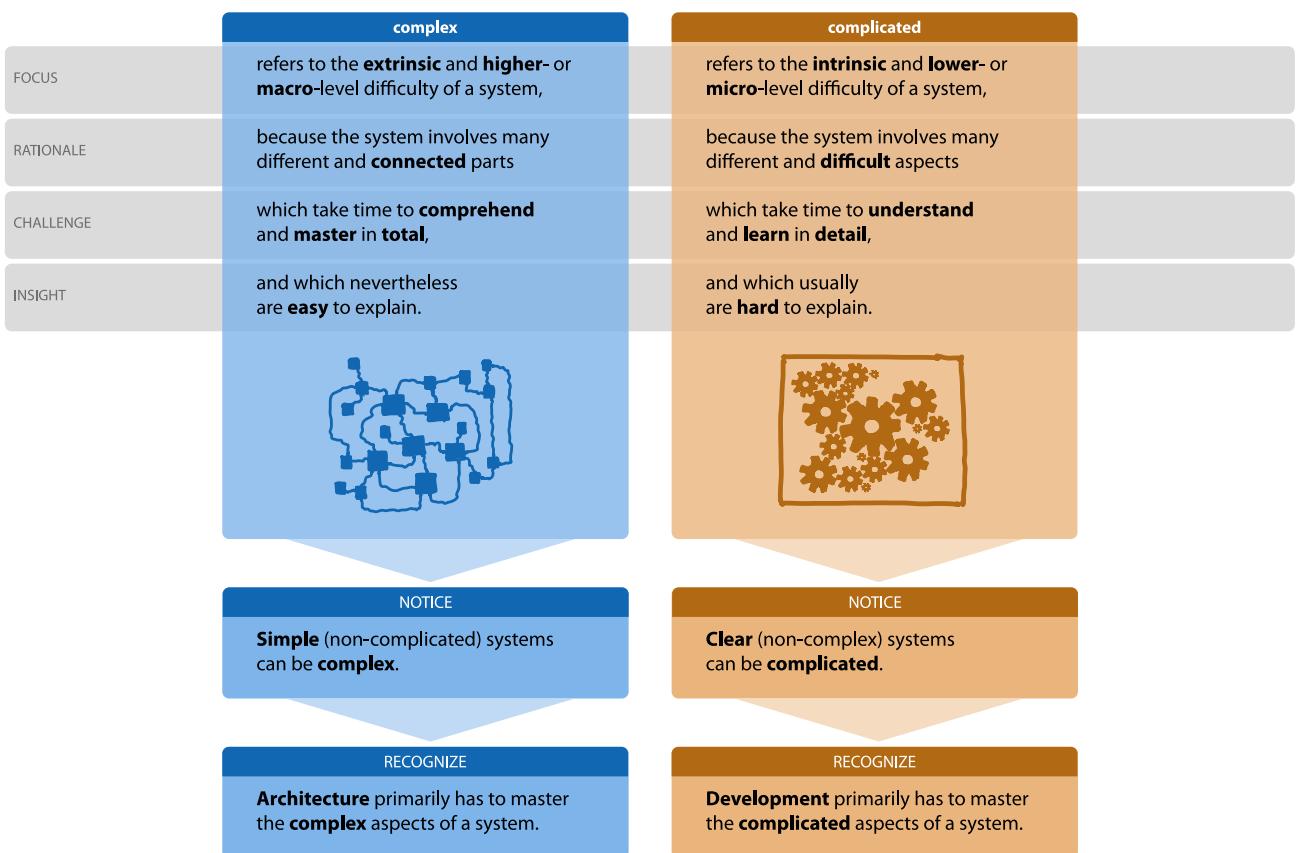
Darüber hinaus gibt es aber zusätzliche Werte, welche ebenfalls eine zentrale Rolle spielen und noch sehr viel mehr geschätzt werden: **Sustainable Concepts** (der Inhalt von **Architecture Fundamentals!**), **Pragmatic Making**, **Constructive Craftsmanship**, **Accredited Creativity**, **Proactive Improvement**, **Inherent Quality** und **Operational Delight**.

## Fragen

?

(keine)

# Complex vs. Complicated



“Komplex” bezieht sich auf die extrinsische Schwierigkeit eines Systems auf der höheren Ebene bzw. der Makro-Ebene, da das System aus vielen verschiedenen, miteinander verbundenen Teilen besteht. Es braucht deshalb Zeit, um dieses zu begreifen und als Ganzes zu meistern, obwohl jedes Teil des Systems meist leicht zu erklären ist.

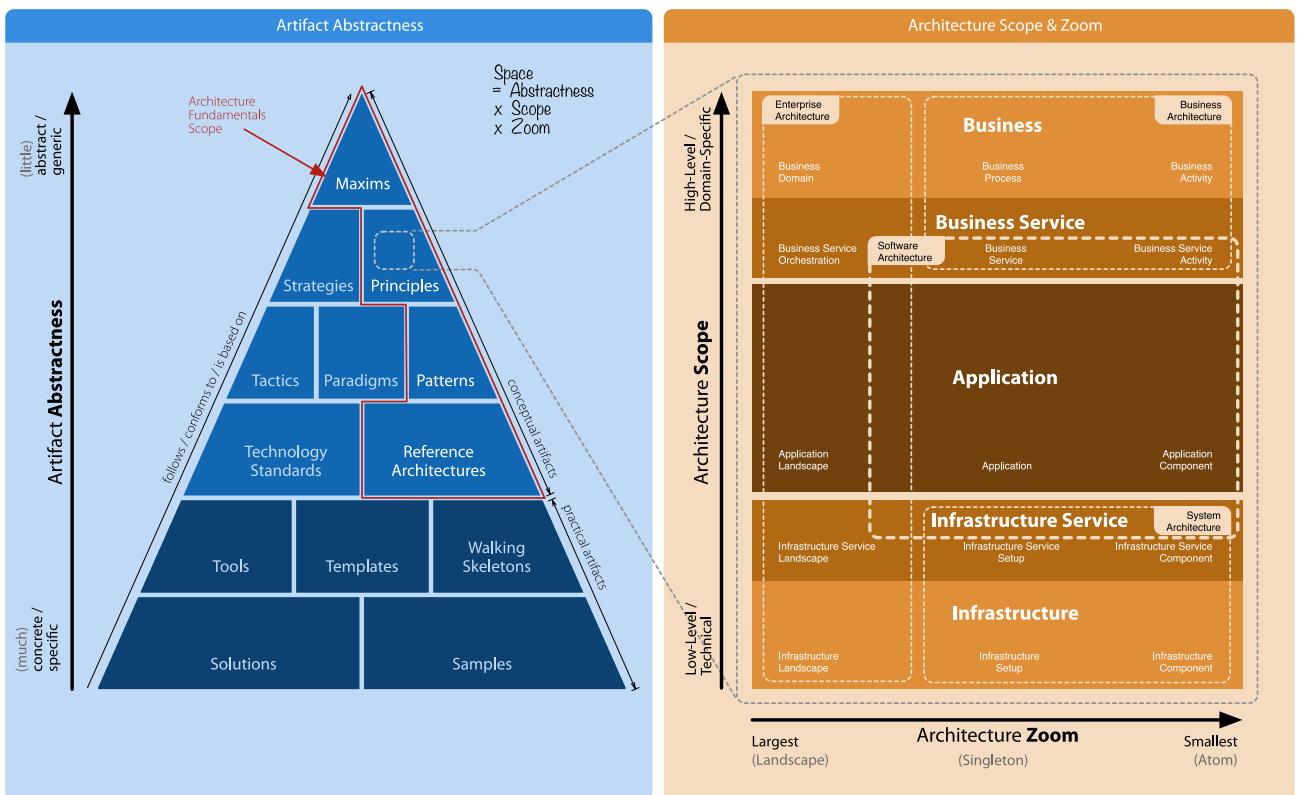
“Kompliziert” bezieht sich auf die intrinsische Schwierigkeit eines Systems auf der unteren Ebene bzw. der Micro-Ebene, da das System viele verschiedene, schwierige Aspekte umfasst. Es braucht deshalb Zeit, um die Teile im Details zu verstehen und zu lernen und jedes Teil des Systems ist meist auch schwer zu erklären.

Beachte: Einfache (nicht-komplizierte) Systeme können komplex sein – offensichtliche (nicht-komplexe) Systeme können kompliziert sein.

Der entscheidende Unterschied ist: Die Architektur bzw. die Konstruktion muss vor allem die komplexen Aspekte eines Systems meistern. Die Entwicklung bzw. die Realisierung muss vor allem die komplizierten Aspekte eines Systems meistern.

## Fragen

- ❓ Muss sich Architektur primär um **komplexe** oder **komplizierte** Aspekte eines Systems kümmern?



Der **IT Architecture Space** besteht aus drei Dimensionen: der **Artifact Abstractness** (einem Abstraktheitsgrad aller Artefakte, die der Architekt kennt), dem **Architecture Scope** (dem Bereich von Architektur, in dem man agiert) und dem **Architecture Zoom** (der Detailstufe von Architektur, in der man agiert).

Man unterscheidet primär drei Architecture Scopes: (high-level/domain-specific) **Business**, **Application** und (low-level/technical) **Infrastructure**. Zusätzlich werden die zwei sekundären Architecture Scopes **Business Service** und **Infrastructure Service** genutzt, um den “gedanklichen Sprung” von Business zu Application und von Application zu Infrastructure in der Praxis deutlich kleiner und sinnvoller ausfallen zu lassen.

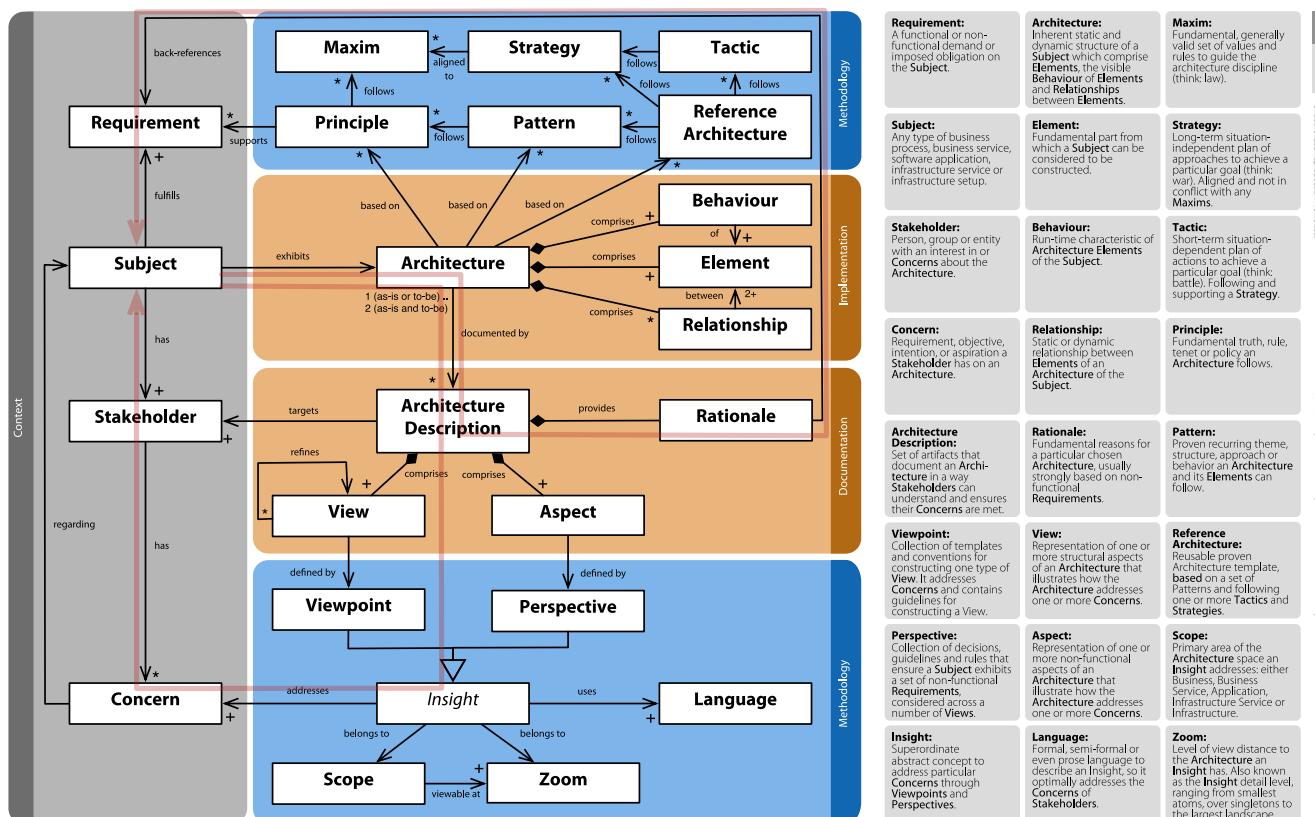
Auf jedem Architecture Scope kann man mindestens drei verschiedene **Architecture Zooms** unterscheiden: von **Landscape** (largest), über **Singleton** bis zu **Atom** (smallest).

Im Raum aus Architecture Scope und Zoom unterscheidet man vier Arten von **IT Architecture**: **Business Architecture** (Facharchitektur), **Software Architecture** (Software-Architektur), **System Architecture** (System-Architektur) und **Enterprise Architecture** (Unternehmens-Architektur).

Umso höher/niedriger der Abstraktheitsgrad der Artefakte, umso größer/kleiner ist üblicherweise die Abdeckung des Raums aus Architecture Scope und Architecture Zoom.

## Fragen

- ?
- Aus welchen drei Dimensionen besteht der IT Architecture Space?
- ?
- Welche vier Arten von Architektur kennt man in der IT?



Damit Architekten sinnvoll in der Praxis kommunizieren können, muss man sich auf ein paar wenige Grundbegriffe und deren Bedeutung einigen. Die Begriffe werden in einer Taxonomie definiert und in der **Architecture Ontology** in Beziehung zueinander gesetzt.

In der Architecture Ontology gibt es vor allem zwei wichtige "Schleifen". Beide starten beim **Subject**, welches eine **Architecture** hat, welche über die **Architecture Description** (im Deutschen "IT-Konzept") beschrieben wird.

**Schleife 1:** Die Architecture Description liefert **Rationales** (Begründungen) für Entscheidungen, welche im Idealfall auf **Requirements** zurück referenzieren sind. Denn eine Architecture Description soll nicht das **WAS** sondern vor allem das **WARUM** dokumentieren. Denn das WAS sieht man auch im Code, aber das WARUM nicht!

**Schleife 2:** Die Architecture Description besteht aus **Views** (Sichten) und **Aspects** (Aspekten), welche methodisch **Viewpoints** und **Perspectives** genannt werden. Beide zusammen liefern **Insights** (Einblicke) auf **Scope** und **Zoom Level** (siehe **Architecture Space**) und werden über eine bestimmte (graphische oder textuelle) **Language** dokumentiert. In jedem Fall werden nur solche Insights gegeben, welche einen **Concern** (Anliegen) eines **Stakeholders** (Interessenvertreter) adressieren. Denn man programmiert auch nichts, was nicht gebraucht wird!

## Fragen

- ?
- Was sollte eine **Architecture Description** (IT-Konzept) neben dem **WAS** vor allem dokumentieren?
- ?
- Was sollte eine **Architecture Description** (IT-Konzept) über **Insights** (Einblicke) vor allem adressieren?

|   |           |   |           |   |           |  |           |
|---|-----------|---|-----------|---|-----------|--|-----------|
| <b>Business Drives</b>  | <b>BD</b> | <b>Component Orientation</b>  | <b>CO</b> | <b>Separation of Business and Technology</b>  | <b>BT</b> | <b>Adequate Description</b>  | <b>AD</b> |
| Trigger and support the business with technological feasibilities, but always understand the business domain and its demands and align your architecture accordingly.                       | \$        | Master complexity in your architecture through stringent bottom-up use of components on all scopes and zoom-levels, loose coupling between and strong cohesion within components. |           | Strictly separate the business, i.e., domain-specific, aspects from the technical ones, i.e., infrastructural, aspects. Furthermore, ensure the explicit visibility of domain concepts. |           | Provide as much stakeholder-directed architecture description as necessary, and as little as possible.   |           |
| <b>Use-Case Driven Design</b>   | <b>UC</b> | <b>Analytical and Creative Act</b>  | <b>AC</b> | <b>Balance Principles Against Requirements</b>  | <b>PR</b> | <b>Insights through Views &amp; Aspects</b>  | <b>VA</b> |
| Design is how it works and runs, so support your customers in their daily work by directly designing your architecture along their domain-specific use-cases.                               |           | Recognize that every good architecture is based on both analytical engineering and creative artistic aspects.   |           | By weighing them against one another, find a reasonable balance between fundamental architecture principles and your particular non-functional requirements.                            |           | Give insights into your architecture through carefully selected stakeholder-directed separate views and aspects. Express each with the most suitable graphical or textual language.                              |           |
| <b>Proven Basis</b>   | <b>PB</b> | <b>Don't Be Too Clever</b>  | <b>TC</b> | <b>Design for Failure Case</b>  | <b>DF</b> | <b>Continuous Compliance</b>   | <b>CC</b> |
| Never start an architecture from scratch. Instead start from proven reference architectures, patterns and templates. Even if, after some iterations, no initial content is left.            |           | Don't be too clever or tricky, both in your higher-level architecture and lower-level design aspects.   |           | Murphy was an architect: everything which can fail will sometime ultimately fail. Hence, already design for the failure case (think: "pessimistic").                                    |           | Continuously check through qualitative inspections and quantitative measurements whether your architecture and the non-functional requirements are followed and do not drift apart.                              |           |
| <b>No Silver Bullet</b>   | <b>SB</b> | <b>Simplicity Trumps</b>  | <b>ST</b> | <b>Design to Change</b>   | <b>DC</b> | <b>Integration-Figure Architect</b>  | <b>IF</b> |
| There is no "one-size-fits-all" architecture, so accept that although you should reuse proven architecture aspects as much as possible, you will always need to individualize your designs. |           | Create solution parts as simple as possible and only as complex as necessary. And remember: simplicity before generality, use before reuse!                                       |           | Time changes everything, so your solution is already legacy at the first day of release. Hence, already design for its change (think: "agile").   |           | Recognize that you, the architect, are the central integrating figure, having to bridge between the business and technology spheres of people.   |           |
| <b>Stepwise Refinement</b>  | <b>SR</b> | <b>Perfect is the Enemy of Good Enough</b>  | <b>GE</b> | <b>Explicit Decisions</b>   | <b>ED</b> | <b>Eat Your Own Dog-Food</b>   | <b>OF</b> |
| Start with the "big picture" and perform a stepwise top-down refinement of your architecture by going from coarse to fine aspects.  |           | Beware of the perfection pitfall and design your architecture only as good as necessary and not as good as ultimately possible.   |           | Record your major architecture decisions and rationales by taking into account and back-referencing the non-functional requirements.  |           | Theory and practice usually differ. Hence it is vital that every architect has good hands-on experience and must both be able to craft the solution and is willing to hypothetically intensively use it himself. |           |

In der IT Architecture folgt man **Architecture Maxims**, welche grundlegende Richtlinien sind. Man kennt 20 Maxime. Der Architekt sollte den Maximen immer folgen und sie nie brechen.

Beachte: **Proven Basis** und **No Silver Bullet** sagen, daß bei einer Architektur immer auf einer bewährten Basis (z.B. eine Referenz-Architektur) aufgesetzt werden muss, es aber gleichzeitig klar sein muss, daß man diese nicht 1:1 verwenden kann, sondern immer erst anpassen muss.

Beachte: **Stepwise Refinement** und **Component Orientation** sagen, daß vom zeitlichen Prozess her (und aus Gründen der Risikominimierung) man immer vom Groben zum Feinen geht, während die Ergebnisse eine stringente Komponenten-Orienterung aufweisen, wo kleine Komponenten hierarchisch in größeren Komponenten integriert sind.

Beachte: während man als IT-Architekt alle anderen Maxime eigentlich nur akzeptieren muss, ist **Simplicity Trumps** von einer anderen Qualität: nichts in der IT ist wirklich einfach. Wenn etwas einfach wirkt, versteht man meist nur noch nicht genügend davon. Oder jemand hat richtig viel investiert, um es einfach wirken zu lassen. **Simplicity Trumps** bedeutet genau dies zu tun: inhärente komplexe Dinge erst wieder einfach werden zu lassen.

## Fragen

- ?
- Aus Gründen der Risikominimierung sollte der IT-Architekt beim **Stepwise Refinement** schrittweise immer wie vorgehen?