ENGINEERING FUNDAMENTALS

# Software Classes

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 01.1

Business

**Custom Software Development** — CSD

Commercial development of **non**-standardised, **fully** individualised, and **non**-reusable **company**-specific software for a **single** customer.

**Standard Software Development** — STD

Commercial development of standardised, **partially** customisable, and **fully** reusable **domain**-specific software for **many** customers.

**Open Source Software Development** — OSS

**Non**-commercial development of standardised, **highly** customisable, and **fully** reusable **generic** software for **many** customers.

## Class: Graphics & Media

target audience: **consumers & enterprises**

### Graphics Editing Application — GEA

Software for editing and rendering graphics in vector and bitmap format.

Examples: Cinema4D, Maya, Blender, After Effects, Illustrator, Inkscape, Scribus, Photoshop, GIMP, etc.

~~CSD~~ STD OSS

### Graphics Animation Engine — GAE

Software for animating the 2D/3D virtual worlds of games and overlays of TV productions.

Examples: Unity, Unreal Engine, CryENGINE, Godot, HUDS, SPX-GC, Holographics, H2R Graphics, etc.

CSD STD ~~OSS~~

### Audio/Video-Processing System — AVS

Software for live-processing and post-production of audio/video based multimedia streams.

Examples: vMix, OBS Studio, VLC, Lossless Cut, Handbrake, Adobe Premiere, FFmpeg, Nimble, etc.

~~CSD~~ STD OSS

## Class: Business & Data

target audience: **consumers & enterprises**

### Office Productivity Application — OPA

Software for productivity in the desktop-based office environment.

Examples: PowerPoint, Excel, Word, Visio, OmniGraffle, LibreOffice, Outlook, XMind, Firefox, Chrome, etc.

~~CSD~~ STD OSS

### Business Information System — BIS

Software for driving business processes through interactive information management.

Examples: Vote, CampS, Mission Control, IPW, KEZ-PSC, TimeSheet, SAP ERP, OpenProject, etc.

CSD STD ~~OSS~~

### Data Management System — DMS

Software for protocol-based storing and retrieving of persistent data.

Examples: NextCloud, PostgreSQL, CockroachDB, Redis, InfluxDB, Neo4J, Tendermind, Gitea, Vault, etc.

~~CSD~~ STD OSS

## Class: Machinery & Network

target audience: **consumers & enterprises**

### Technical Control System — TCS

Software for controlling a physical machinery or technical system.

Examples: AquaTherm, AVM! FritzBox Firmware, BirdDog Camera Firmware, etc.

CSD ~~STD~~ ~~OSS~~

### Network Communication System — NCS

Software for protocol-based communication of data over a computer network.

Examples: Apache, NGINX, HAProxy, Mosquitto, RabbitMQ, Node-RED, KeyCloak, etc.

~~CSD~~ ~~STD~~ OSS

### Operating System Kernel — OSK

Software kernel for low-level operating a physical or virtual device and run programs on it.

Examples: Windows, macOS, iOS, Linux, FreeBSD, QNX, ChibiOS/RT, Kubernetes, Wildfly, etc.

~~CSD~~ STD OSS

## Class: Development & Tools

target audience: **vendors & suppliers**

### Software Development Kit — SDK

Software libraries and frameworks of reusable functionality for developing software.

Examples: NDI SDK, HAPI, GraphQL-IO, Sequelize, JDK, Spring, Hibernate, etc.

~~CSD~~ ~~STD~~ OSS

### Software Development Tools — SDT

Software tools for editing, linting, compiling, packaging, distributing, and installing software.

Examples: Visual Studio Code, Sublime Text, GCC, GNU Binutils, NPM, JDK, Docker, Helm, etc.

~~CSD~~ ~~STD~~ OSS

### Operating System Tools — OST

Software tools for high-level operating a physical or virtual computing device.

Examples: Coreutils, Bash, Vim, TMux, FZF, cURL, RSYNC, OpenSSH, etc.

~~CSD~~ ~~STD~~ OSS

*audience & deliverable*

Intellectual Content: Version 1.1 (2022-12-10), Authored 2021-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0 (2022-10-29), Copyright © 2021-2022 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

# Software Development Approaches

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 01.2

## Development Approaches

### Software Prototyping *mocking* SP

Develop an early sample or model of a software solution by mocking and cheating in order to just once test a concept, idea or process.



Example: Customer Sales Demo

### Software Bricolage *integrating* SB

Develop a single instance of a software solution by tinkering, cobbling and integrating partial solutions in order to prove feasibility or just provide a service.



Example: Company-Internal SaaS

### Software Craftsmanship *crafting* SC

Develop a production-grade software solution by professional, clean but plain craftsmanship means in order to solve a usually complicated problem.



Example: Open Source Framework

### Software Engineering *teaming* SE

Develop a production-grade software solution by a professional, risk-hedged engineering approach in order to solve a usually complex problem.



Example: Business Information System

### Continuum & Process

The four development approaches do *not* form a hierarchy, but can be combined in practice: **Prototyping** and **Bricolage** can be earlier stages of **Craftsmanship** or **Engineering**. **Craftsmanship** can be part of **Bricolage** or **Engineering**. Each approach requires a special skill (mocking, integrating, crafting, teaming).

## Development Approaches: **Characteristics Comparison** *

| | Effort: Person-Days | Effort: Persons | Process: Risk-Hedge | Process: Traceability | Solution: Target Technology | Solution: Production-Grade | Solution: Sustainability | Solution: Claim | Solution: Life-Time Months | Solution: Lines of Code (k) |
|---|---|---|---|---|---|---|---|---|---|---|
| **Software Prototyping** | 1-20 | 1-2 | - | - | - | - | - | 5% | 0-3 | 0-3 |
| **Software Bricolage** | 5-100 | 1-2 | - | - | x | (x) | - | 60% | 3-24 | 1-10 |
| **Software Craftsmanship** | 5-100 | 1-2 | - | - | x | x | x | 100% | 24-48 | 5-25 |
| **Software Engineering** | >150 | 5-50 | x | x | x | x | x | 80% | >48 | >25 |

* All figures are just rough orders of magnitude for indication and illustration purposes.

### Key Message

All four approaches are equally essential in practice. Which one(s) to choose, entirely depends on the particular requirements.

## Development Approaches: **Success Patterns**

| | **Software Prototyping** | **Software Bricolage** | **Software Craftsmanship** | **Software Engineering** |
|---|---|---|---|---|
| Performance<br>Responsibility<br>Model | One-Man-Show<br>Single<br>Mental | One-Man-Show<br>Single<br>Mental | One-Man-Show<br>Single<br>Mental/**Documented** | **Team Play**<br>**Separated**<br>**Documented** |
| Decisions<br>Process<br>Optimisation | Implicit<br>**Minimized**<br>**Time** | Implicit<br>**Partial**<br>**Efficiency** | Implicit/**Explicit**<br>**Partial**<br>**Effectiveness** | **Explicit**<br>Complete<br>**Economics** |
| Risks<br>Stakeholders<br>Mastering | Ignore<br>Ignore<br>**Time-Constraint** | Ignore<br>Ignore<br>Complexity | Ignore<br>Ignore<br>**Complication** | **Mitigate**<br>**Manage**<br>Complexity |
| Solutions<br>Standards<br>Efforts | **Use Full**<br>Use<br>**Configuration** | Use Partial<br>Use<br>**Integration** | Use Partial<br>**Potentially Create**<br>Programming | Use Partial<br>Use<br>Programming |
| Target<br>Sustainability<br>Traceability | **Demo**<br>**No**<br>**No** | Solution<br>**Partial**<br>**No** | **Product**<br>Full<br>**Partial** | **Product**<br>Full<br>Full |

Intellectual Content: Version 1.04 (2022-11-10), Authored 2021-2022 by Dr. Ralf S. Engelschall (Graphical Illustration: Version 1.03 (2021-10-25)) Copyright © 2021 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved. Unauthorized reproduction, modification and redistribution is not permitted. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

goal & approach

ENGINEERING FUNDAMENTALS

# Software Engineering

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 01.3

Intellectual Content: Version 1.0.2 (2021-09-06), Authored 2021 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.2 (2021-09-11), Copyright © 2021 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

## Engineering

### Machinery

The collection of physical components of a mechanical apparatus using electrical power to perform a certain task.

created by

### Engineering

The systematic application of **scientific and technological knowledge, principles, approaches, practices, methods, and experiences**, to **design, build, operate, and maintain** complex production-grade solutions through a **traceable** process. *

taking into account

### Engineering Code of Conduct

Commit yourself to the **highest ethical and professional conduct**, accept responsibility in making decisions consistent with the **safety, health, and welfare of the public**, apply the **state of the art in science and technology** of your field, perform services only in areas of your **competence**, and comply with the principles of **sustainable development**. *

* derived from definitions and Code of Ethics statements of ASCE, NSPE, ACM, IEEE, etc.

specialised by

### Hardware

The collection of physical components (machinery) of a computer system.

operated by

## Software Engineering

### Software

The collection of instructions and data that tell a computer system how to work.

created by

### Software Engineering (transferred)

The systematic application of scientific and technological knowledge, principles, approaches, practices, methods, and experiences, to design, build, operate, and maintain complex production-grade **software** solutions through a traceable process.

simplified to

### Software Engineering (simplified)

The systematic application of **engineering** knowledge, principles, approaches, practices, methods, and experiences to **the development of software.**

also taking into account

transferred to

### Wording

**Engineering**: derived from the *Latin* words **ingeniare** (to create, generate, contrive, devise) and **ingenium** (cleverness).

*profession & ethics*

ENGINEERING FUNDAMENTALS

# Profession Characteristics

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 01.4

Intellectual Content: Version 1.00 (2024-06-10), Authored 2024 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.00 (2024-06-10) Copyright © 2024 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

| | creative | rational |
|---|---|---|
| | involving the use of **strong imagination** and **original ideas** | involving the use of **analytical thinking** and **logical reasoning** |
| practical — involving the **actual doing** and **real application** of something | **CRAFTSMAN** (Handwerker)<br>Objectives: **solve problems**, **create uniques**, …<br>Software Engineering: **Programming**, **Configuration**, … | **ENGINEER** (Ingenieur)<br>Objectives: **solve problems**, **create solutions**, …<br>Software Engineering: **Estimation**, **Automation**, … |
| theoretical — involving the **manifested imagination** or **analytical study** of something | **ARTIST** (Künstler)<br>Objectives: **create beauty**, **create uniques**, …<br>Software Engineering: **Visual Design**, **Coding Style**, … | **SCIENTIST** (Wissenschaftler)<br>Objectives: **analyse problems**, **create knowledge**, …<br>Software Engineering: **Methodology**, **User Research**, … |

**INVENTOR** (Erfinder)

diagram.direct / XIRF-NIIV

ENGINEERING FUNDAMENTALS

# TRUE Manifesto

true-manifesto.org

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 01.5



## targeted

adequate
suitable
focused

Statement: **We focus on adequate and suitable solutions and approaches.**

Rationale: **Both solutions and approaches have to be in a reasonable proportion to the problem.**

Implications:
- We avoid both over-engineered and cobbled-together solutions.
- We avoid "one-size-fits-all" approaches.
- We suitably adapt solutions, tools and methods.



## reasoned

considered
assessed
deliberate

Statement: **We think carefully and holistically in advance about our solutions and approaches.**

Rationale: **We always think large, even if we have to act small, because thinking in advance is more efficient and effective than correcting afterwards.**

Implications:
- We always develop the "big picture" first and add ancillary details as late as possible.
- We are opinionated and steadfast regarding our decisions and solutions.
- We know that conceptual modeling is key to understanding both problems and solutions.



## up-to-date

educated
experienced
insistent

Statement: **We develop high-quality solutions on the basis of up-to-date methods and technologies.**

Rationale: **We have to cope with the fact that the IT world is recurrently revolutionizing itself.**

Implications:
- We continuously educate ourselves.
- We continuously and critically challenge and assess emerging approaches and products.
- We are not satisfied with mediocre solutions.



## evolutionary

sustainable
harmonic
contextual

Statement: **We develop sustainable solutions that optimally fit into their context.**

Rationale: **Nature teaches us that only evolutionary approaches and solutions have a good chance to survive in the long run.**

Implications:
- We actively learn from experiences of the past in order to improve the future.
- We avoid "quick hacks", as they are not long-term solutions, but just short-term means to get rid of problems.
- We assure that our solutions can be reasonably maintained in the long-term.

Intellectual Content: Version 1.0.7 (2021-10-25) Authored 2019-2021 by Dr. Ralf S. Engelschall, Christian Reiser, Rudolf Koster and Matthias Buddeyins
Graphical Illustration: Version 1.0.7 (2021-10-25) Copyright © 2019-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

*entitlement & values*

ENGINEERING FUNDAMENTALS

# Discipline Claim

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 01.6

## FA FARSIGHTED
weitblickend

**Be farsighted in your solution finding.**

Sei weitblickend in deiner Lösungsfindung.

AR: Scalable Hub'n'Spoke
DV: Plugin SPI



## TE TENET-ORIENTED
grundsatzorientiert

**Orientate yourself on fixed tenets in your approach and solution finding.**

Orientiere dich an festen Grundsätzen in deinem Vorgehen und deiner Lösungsfindung.

AR: Separation of Concern
DV: Strict Coding-Style



## TH THOUGHTFUL
wohlüberlegt

**Act thoughtful in your approach and solution finding.**

Agiere wohlüberlegt in deinem Vorgehen und deiner Lösungsfindung.

AR: Modularization
DV: Algorithmical Control Structure



## HO HOLISTICALLY
ganzheitlich

**Think holistically and in the long-term when finding your solutions.**

Denke ganzheitlich und langfristig in deiner Lösungsfindung.

AR: Walking Skeleton Design
DV: Consistent Error Handling



## AD ADEQUATE
angemessen

**Ensure that your approach and solutions are adequate to the boundary conditions.**

Sorge dafür, daß dein Vorgehen und deine Lösungen angemessen zu den Rahmenbedingungen sind.

AR: No Cloud-Native Complexity
DV: No Over-Engineered Abstractions



## FE FEASIBLE
machbar

**Ensure that your approach and solutions can be realised at reasonable costs.**

Sorge dafür, daß dein Vorgehen und deine Lösungen mit vernünftigen Kosten realisiert werden können.

AR: Existing Framework Functionality
DV: Realistic Programming Model

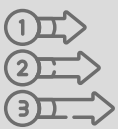


## IN INCREMENTAL
inkrementell

**Apply the depth of your discipline incrementally.**

Wende die Tiefe deiner Disziplin inkrementell an.

AR: Identified Solution Cruxes
DV: Minimum Viable Product




## VA VALUEABLE
wertvoll

**Provide clearly recognizable added values with your approach and solutions.**

Liefere klar ersichtliche Mehrwerte mit deinem Vorgehen und deinen Lösungen.

AR: Technology Stack Design
DV: User-Story-Driven Functionality



## SU SUSTAINABLE
nachhaltig

**Create sustainable solutions that are well integrated into their environment.**

Erschaffe nachhaltige Lösungen, die gut in ihre Umgebung integriert sind.

AR: Interoperable Interfaces
DV: Maintainable Code



Intellectual Content: Version 1.0.1 (2024-06-17), Authored 2023-2024 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.1 (2024-06-17), Copyright © 2023-2024 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

# Sustainability

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 01.7

## HI Minimize HARDWARE Idleness

**Minimize the idleness and maximize the utilization of existing hardware resources.**

**Rationale:** Unused or under-utilized hardware are an unnecessary waste of already available resources.

**Keywords:** Virtualization, Utilization.



## DE Minimize DESIGN Excessiveness

**Minimize the excessiveness and maximize the adequacy of solution designs.**

**Rationale:** Non-adequate designs cause unnecessary complexity and waste resources.

**Keywords:** Reduced Libraries, Immutability.



## HE Minimize HUMAN Effort

**Minimize the efforts of humans and maximize the efforts of machines in all production and operation processes.**

**Rationale:** Delegating tasks to machines gives humans the possibility to concentrate on more important tasks.

**Keywords:** Computer, Robot, Automation.



## SI Minimize SOFTWARE Inefficiency

**Minimize the inefficiency and maximize the efficiency of software applications and their development processes.**

**Rationale:** Efficient software and development processes consume less resources.

**Keywords:** Caching, Monolith.



## SE Minimize SOLUTION Ephemerality

**Minimize the ephemerality and maximize the life-span of any type of solutions.**

**Rationale:** Short life-spans of solutions cause unnecessary short renewals and this way wastes resources.

**Keywords:** High Quality, Best Practice.



## EC Minimize ENERGY Consumption

**Minimize the consumption and maximize the saving of energy in all production and operation processes.**

**Rationale:** Electric energy still has to be partially generated from non-renewable resources.

**Keywords:** Eco Mode, Reduced CI/CD.



## IA Minimize INFORMATION Amount

**Minimize the total amount of gathered, transmitted, stored and spreaded information.**

**Rationale:** Reduced amount of information means less data transmission, less data storage, less GDPR issues, etc.

**Keywords:** Compression, No Big Data.



## EE Minimize ECOSYSTEM Exploitation

**Minimize the exploitation and maximize the back-contribution in any type of ecosystems.**

**Rationale:** The consumer and provider behaviour have to be in balance for every long-lasting ecosystem.

**Keywords:** Open Source Software.



## CE Minimize CARBON Emission

**Minimize the carbon emission and hence the footprint during any type of production and operation processes.**

**Rationale:** Climate change and global warming is partially caused or at least accelerated by carbon emissions.

**Keywords:** Reduced CO2 Footprint.



Intellectual Content: Version 1.0.0 (2022-10-31) Authored 2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.0 (2022-10-31) Copyright © 2022 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

# Software Engineering Metamodel

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 02.1

Intellectual Content: Version 1.04 (2021-09-16), Authored 2021 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.2 (2021-10-11), Copyright © 2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

## Craftsmanship Model 4

WHEREBY & HOW

Know how in a profession to operationally perform certain practices with the help of templates and tools.

**Profession**
A person's occupation, founded upon specialized educational training and formal qualification.

**Know-How**
Distinct overall knowledge on how to perform certain tasks.

**Practice**
Distinct steps to perform a certain task with the help of a certain know-how and template.

**Template**
The initial version of an artifact in order to speed up and/or support crafting the artifact.

**Tool**
Software application to create an artifact from scratch or from a template and further edit it.

## Discipline Model 1

HOW & WHAT

Based on personal inclinations, understand how to act in area-grouped disciplines and create artifacts and their aspects.

**Inclination**
A person's natural tendency or urge to act or feel in a certain way.

**Area**
Logical group of strongly related disciplines.

**Discipline**
Distinct knowledge, responsibility and objectives within Software Engineering.

**Artifact**
Single representation of an internally used and/or externally delivered result.

**Aspect**
Smallest result unit of an artifact which can be created separately.

## Workflow Model 2

WHAT & WHO

Understand the workflow of separate iterating cycles and work streams, comprised of one or more steps.

**Workflow**
The five distinct iteration cycles Product Management, Business, Development, Operations and Project Management.

**Cycle**
Logical iteration cycle of a sequence of strongly related and dependent workflow steps.

**Step**
Single step in a workflow cycle with a clear logical previous and next step in the logical work flow.

**Flow**
Logical sequence of interweaved, inter-cycle, optionally skipped, and dependent steps.

**Key Message**
Mastering Software Engineering means to understanding (in this order) **Discipline**, **Workflow**, **Process** and **Craftsmanship** models. Especially, *never* start with a particular Process model, as it is secondary!

## Process Model 3

WHO & WHEN

Map activities onto a project execution schedule, based on horizontal team focus tracks and vertical temporal project periods and phases.

**Role**
Distinct responsibility and goal of an individual person.

**Track**
Work stream consisting of a continuously sequence of steps of a workflow cycle.

**Activity**
Distinct activity in the process to perform a certain task or complete a particular work-package.

**Period**
Minor time period in the process, to split the project phases into distinct time-, cost-, or scope-based time units.

**Phase**
Major time period in the process, to split the project into distinct focus periods, separated by major milestones.

+ suitable for
+ correlates to
serialized by 1
+ operationalised by
+ corresponds to
operationalises 1
+ derived from

*big picture & coherency*

ENGINEERING FUNDAMENTALS

# Software Engineering Disciplines

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 02.2

## ANALYSIS AN

### Software Requirements REQ

**Identify Needs:** 3 BB
We understand which outcomes of the solution are most valuable to users.

*Requirements Engineer / Business Analyst*

### Domain Modeling DOM

**Determine Solution:** 2 WB
We model and specify the solution through involved functional and non-functional aspects.

*Business Analyst / Business Architect*

## ARCHITECTURE AR

### Software Architecture SWA

**Design Software:** 1 WB
We design an orthogonal, well-balanced and well-considered solution.

*Software Architect*

### System Architecture SYA

**Design Systems:** 2 BB
We ensure that the solution fits optimally into its environment.

*System Architect / Enterprise Architect*

## CONFIGURATION CF

### Software Versioning VER

**Version Artifacts:** 1 BB
We place every artifact of the solution under strict version control.

*Configuration Manager*

### Software Assembly ASM

**Assemble Artifacts:** 1 BB
We build and package the solution through an automated and repeatable mechanism.

*Build Manager / Build Engineer*

## ANALYTICS AC

### Software Reviewing REV

**Review Code:** 4 WB
We regularly and semantically peer-review the source code of the solution.

*Software Tester*

### Software Testing TST

**Test Solution:** 2 BB
We adequately test the functional and non-functional aspects of the solution.

*Software Tester*

## MANAGEMENT MG

### Product Management PRD

**Push Product:** 3 BB
We continuously push the development and release of the solution to the users.

*Product Manager / Product Owner*

### Project Management PRJ

**Steer Process:** 3 BB
We rigorously balance time, cost and scope to react on changes and reach the goals.

*Project Manager*

**business-oriented** & domain-specific | **constructive** & technological | **infrastructural** & technological | **analytical** & domain-specific | **people-oriented** & process-oriented

## EXPERIENCE EX

### User Experience UXP

**Optimize Workflows:** 3 BB
We align the solution to the perspective of the target audience.

*User Experience Expert*

### User Interface UID

**Design User Interfaces:** 2 WB
We design a useful, intuitive, and beautiful user interface for the solution.

*User Interface Designer / Graphics Designer*

## DEVELOPMENT DV

### Software Development DEV

**Implement Code:** 1 WB
We develop the solution outside-in, from coarse to fine aspects.

*Software Engineer / Software Developer*

### Software Refactoring REF

**Refactor Code:** 4 BB
We regularly and holistically refactor the solution to ensure long-term quality.

*Software Engineer / Software Developer*

## DELIVERY DL

### Software Deployment DPL

**Deploy Artifacts:** 1 BB
We ship and deploy the solution through an automated and repeatable mechanism.

*System Engineer*

### System Operations OPS

**Operate Solution:** 4 BB
We ensure that our infrastructures and the solution can be operated in a resilient and secure way.

*System Administrator / System Operator*

## COMPREHENSION CP

### Usage Documentation DOC

**Document Solution:** 2 WB
We adequately document the usage and operation of the solution.

*Technical Writer*

### User Training TRN

**Train Users:** 4 BB
We adequately train the users and operators of the solution.

*Product Expert*

## ADJUSTMENT AD

### Project Coaching COA

**Support Members:** 4 BB
We ensure that project members use state-of-the-art methodology, technology, and tools.

*Project Coach / Methodology Master*

### Change Management CHG

**Involve Stakeholders:** 3 BB
We ensure that all stakeholders of the solution are suitably involved.

*Change Manager*

WB white-box view (details before whole) | BB black-box view (whole before details) | X scalability layer (from 4/most to 1/least dispensable)

*inclination & knowledge*

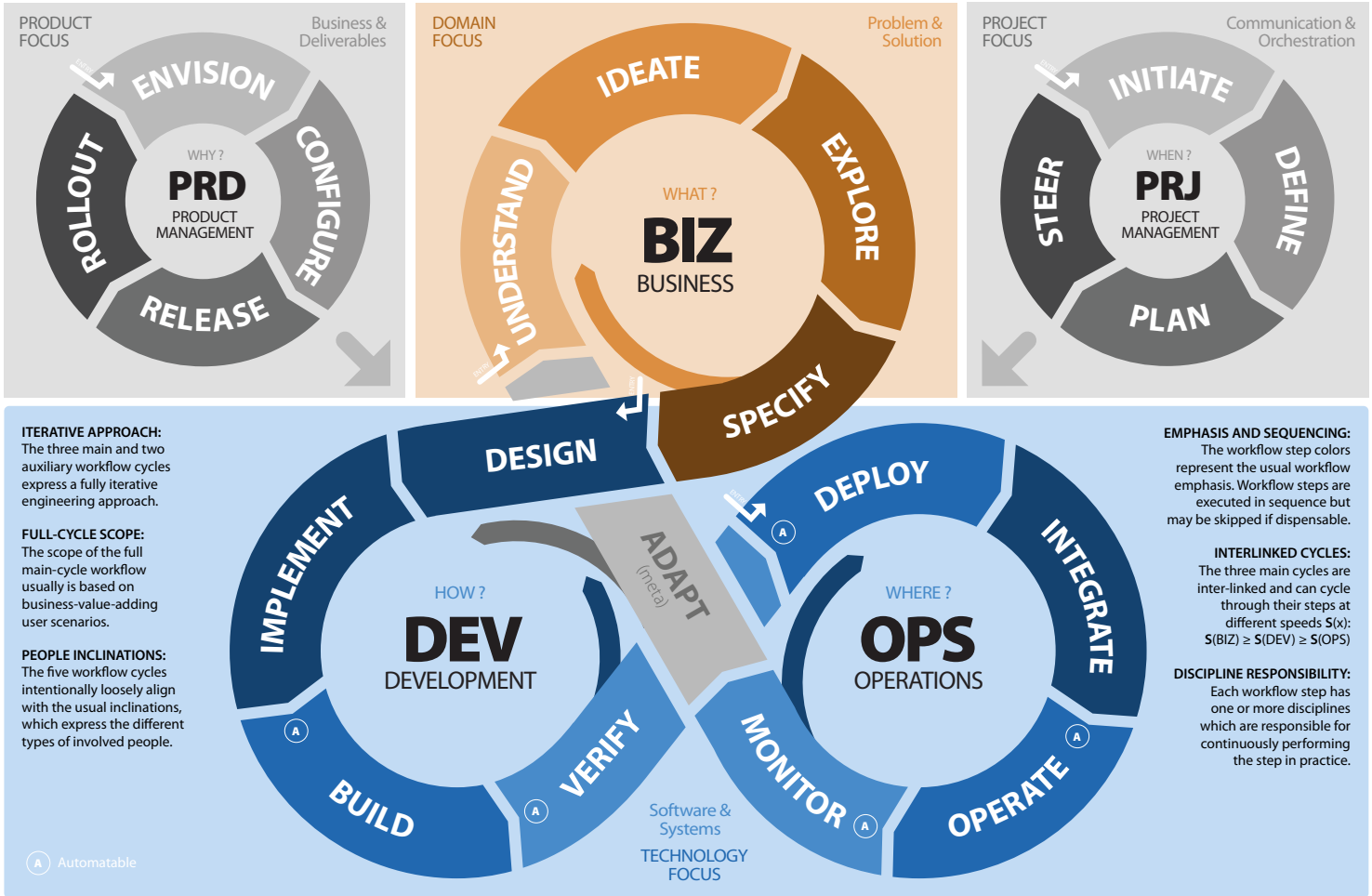
Intellectual Content: Version 3.0.9 (2022-12-10), Authored 2009-2022 by Dr. Ralf S. Engelschall and Rudolf Koster
Graphical Illustration: Version 3.0.2 (2021-09-11), Copyright © 2009-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

# Software Engineering Workflow

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 02.3




**ITERATIVE APPROACH:**
The three main and two auxiliary workflow cycles express a fully iterative engineering approach.

**FULL-CYCLE SCOPE:**
The scope of the full main-cycle workflow usually is based on business-value-adding user scenarios.

**PEOPLE INCLINATIONS:**
The five workflow cycles intentionally loosely align with the usual inclinations, which express the different types of involved people.

A Automatable

**EMPHASIS AND SEQUENCING:**
The workflow step colors represent the usual workflow emphasis. Workflow steps are executed in sequence but may be skipped if dispensable.

**INTERLINKED CYCLES:**
The three main cycles are inter-linked and can cycle through their steps at different speeds **S**(x): **S**(BIZ) ≥ **S**(DEV) ≥ **S**(OPS)

**DISCIPLINE RESPONSIBILITY:**
Each workflow step has one or more disciplines which are responsible for continuously performing the step in practice.

Intellectual Content: Version 1.1.1 (2021-11-29), Authored 2019-2021 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0 (2021-09-12), Copyright © 2020-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contents only

focus & steps

ENGINEERING FUNDAMENTALS

# Software Engineering Steps

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 02.4

## PRODUCT MANAGEMENT — PRD

### ENVISION
PRD, UXP

**Envision Solution:**
We envision the solution functionality and quality from the business and user perspectives.

### CONFIGURE
PRD, VER

**Configure Version:**
We configure versions of the solution from versioned artifacts and their feature-sets.

### RELEASE
PRD, ASM, DOC

**Release Version:**
We create and release a distinct version of the solution.

### ROLLOUT
CHG, TRN

**Rollout Version:**
We adequately inform, involve and train the users and operators of the solution.

## BUSINESS EFFORTS — BIZ

### UNDERSTAND
REQ, UXP, PRD

**Understand Problem:**
We empathically understand the problem and requirements of the users.

### IDEATE
UXP, REQ

**Ideate Solution:**
We find an adequate solution for the problem and the requirements of the users.

### EXPLORE
UXP, UID, SWA, DEV

**Explore Ideas:**
We prototype, explore, and assess ideas, approaches and technologies for the solution.

### SPECIFY
DOM, UID

**Specify Solution:**
We rigorously and completely specify the functionality and quality of the solution.

## DEVELOPMENT EFFORTS — DEV

### DESIGN
SWA, SYA

**Design Architecture:**
We design how to implement the solution in an orthogonal, adequate and sustainable way.

### IMPLEMENT
DEV, DOC, REF, REV

**Implement Solution:**
We implement the solution outside-in, from coarse to fine aspects.

### BUILD
ASM, VER

**Build Artifacts:**
We build and package the solution from versioned artifacts.

### VERIFY
TST, REV

**Verify Solution:**
We rigorously, but adequately, review and test the functional and non-functional aspects of the solution.

## OPERATIONS EFFORTS — OPS

### DEPLOY
DPL, OPS, SYA

**Deploy Artifacts:**
We ship and deploy the solution releases and their updates in an automated and repeatable way.

### INTEGRATE
DPL, OPS, SYA

**Integrate Environment:**
We integrate the solution with its target environment.

### OPERATE
OPS

**Operate Solution:**
We ensure that our infrastructures and the solution can be operated in a resilient and secure manner.

### MONITOR
OPS, TST

**Monitor Solution:**
We continuously monitor our infrastructures and the solution under run-time.

## PROJECT MANAGEMENT — PRJ

### INITIATE
PRJ, COA

**Initiate Project:**
We initially setup the project on the contract and resource level.

### DEFINE
PRJ, PRD

**Define Constraints:**
We define the constraints of the project on the time, cost, and scope level.

### PLAN
PRJ, PRD

**Plan Tasks:**
We continuously plan the next iterations, their steps and their tasks in the project.

### STEER
PRJ

**Steer People:**
We rigorously and continuously balance time, cost and scope to react on changes and still fulfilling the constraints.

steps & goals

Intellectual Content: Version 1.0.1 (2021-09-13) Authored 2021 by Dr. Ralf S. Engelschall and Rudolf Koster
Graphical Illustration: Version 1.0.1 (2021-09-13) Copyright © 2021 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

# Software Engineering Process

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 02.5

**1. WORKFLOW CYCLES**
The workflow has five cycles which continuously iterate through their steps. Workflow steps are executed in each cycle in sequence, but may be skipped if dispensable in a particular iteration of the process. The length of an iteration is arbitrary, but can be e.g. about 1/3 of a Scrum sprint.

**2. WORKFLOW STEPS:**
The workflow steps describe a logical activity which has to be performed. Each step relates to one or more discipline areas and their corresponding disciplines, which express the operative responsibilities for each workflow step. In each discipline individual roles act.

**3. WORKFLOW ROLES:**
The workflow roles are held by individual persons. Each role is primarily responsible for a particular workflow step. In addition, each role can be secondarily responsible for other workflow steps or at least actively support those steps.

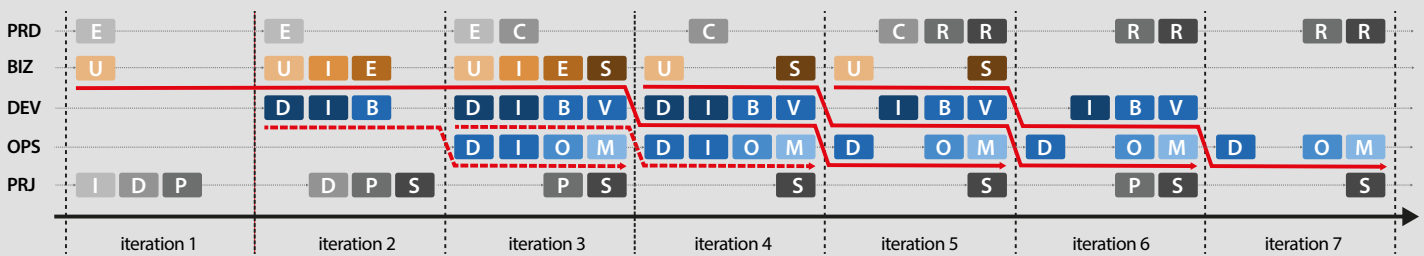
**4. PROJECT SCHEDULE:**
To create a particular project execution schedule, the five cycles, their iterations and their steps have to be mapped onto a timeline. The cycles are mapped onto (horizontal) timeline tracks, the iterations are mapped onto (vertical) timeline phases, and the steps are mapped onto timeline activities.

**5. PROCESS FLOWS (THE CRUX):**
The activities across the cycles can (and should) be linked into individual (diagonal) waterfall-like flows, although the execution schedule, from the perspective of the cycles, is fully iterative. There are multiple such flows in parallel and they are usually highly interleaved on the project timeline in order to maximally utilize the team.

**6. PROCESS ADAPTION:**
In the meta-step ADAPT, the process is adapted by choosing which workflow steps are required for the next iteration. The major input for this decision is the current solution state and the feedback on it by the customer.

| | | business-oriented & domain-specific | | | | constructive & technological | | | | infrastructural & technological | | | | analytical & domain-specific | | | | people-oriented & process-oriented | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | AN | | EX | | AR | | DV | | CF | | DL | | AC | | CP | | MG | | AD | |
| | | REQ | DOM | UXP | UID | SWA | SYA | DEV | REF | VER | ASM | DPL | OPS | REV | TST | DOC | TRN | PRD | PRJ | COA | CGH |
| | | Requirements Engineer | Business Architect | User Experience Expert | User Interface Designer | Software Architect | System Architect | Software Developer | Software Developer | Configuration Manager | Build Manager | System Engineer | System Administrator | Software Tester | Software Tester | Technical Writer | Product Trainer | Product Owner | Project Manager | Project Coach | Change Manager |
| PRD | ENVISION | + | + | * | | | | | | | | | | | | | | * | | | |
| | CONFIGURE | + | + | + | | | | | | * | | | | | | | | * | | | |
| | RELEASE | | | | | | | | | + | * | | | | | * | + | * | + | | + |
| | ROLLOUT | | | | | | | | | | | | | | | | * | + | + | | * |
| BIZ | UNDERSTAND | * | * | * | | | | | | | | | | | | | | * | | | |
| | IDEATE | + | + | * | | | | | | | | | | | | | | + | | | |
| | EXPLORE | + | + | * | * | * | + | * | | | | | | | | | | + | | | |
| | SPECIFY | * | * | + | * | + | + | | | | | | | | + | + | | + | | | |
| DEV | DESIGN | + | + | + | + | * | * | | | + | + | + | | + | + | | | + | | | |
| | IMPLEMENT | | | | * | + | + | * | * | + | + | | | * | | * | | | | | |
| | BUILD | | | | | + | + | + | | * | * | + | | + | | | | | | | |
| | VERIFY | + | + | + | + | + | + | + | | | | | | * | * | | | + | | | |
| OPS | DEPLOY | | | | | + | * | + | | + | + | * | * | | | | | | | | + |
| | INTEGRATE | | | | | + | * | | | + | + | * | * | | | | | | | | + |
| | OPERATE | | | | | | | | | | | + | * | | | | | | | | |
| | MONITOR | + | + | + | | + | + | | | | | + | * | | * | | | + | | | |
| | ADAPT | + | + | + | | + | + | | | | | | | | | | | * | * | * | + |
| PRJ | INITIATE | | | | | | | | | | | | | | | | | | * | * | |
| | DEFINE | + | + | | | + | | | | | | | | | | | | * | * | * | |
| | PLAN | + | + | | | + | | | | | | | | | | | | * | * | | |
| | STEER | + | + | | | + | | | | | | | | | | | | | * | | |

* responsible (primarily)
* responsible (secondarily)
\+ supporting




roles & tasks

Intellectual Content: Version 1.0.9 (2025-10-18) Authored 2019-2021 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.1.1 (2021-09-11) Copyright © 2020-2021 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited
Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only

ENGINEERING FUNDAMENTALS

# Software Engineering Artifacts

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 02.6

## 1 Software **Requirements** Specification *input / what* REQ

| | |
|---|---|
| Requirements: Customer Journey — REQ UXP PRD — 2 ENVISION | Domain Model: Personas — REQ DOM UXP — 3 UNDERSTAND |
| Requirements: Solution Vision — REQ UXP PRD — 2 ENVISION | Domain Model: Test Cases — REQ DOM TST — 3 SPECIFY |
| Requirements: Functional Requirements — PRD UXP REQ — 1 UNDERSTAND | User Interface: Usage Concept — UXP UID — 2 SPECIFY |
| Requirements: Non-Functional Requirem. — PRD SWA REQ — 1 UNDERSTAND | User Interface: Language Conventions — REQ UXP UID — 3 SPECIFY |
| Domain Model: Data Model — REQ DOM — 1 SPECIFY | User Interface: Dialog Patterns — UXP UID — 3 SPECIFY |
| Domain Model: Use Cases — UXP REQ DOM — 1 SPECIFY | User Interface: Dialog Storyboard — REQ UXP UID — 1 SPECIFY |
| Domain Model: Use Case Scenarios — UXP REQ DOM — 2 SPECIFY | User Interface: Visual Design — PRD UXP UID — 3 SPECIFY |

## 2 Software **Architecture** Specification *input / how* ARC

| | |
|---|---|
| Viewpoint: Context View — SYA SWA PRD — 2 ENVISION | Perspective: Configurability & Extensibili. — DEV SWA DPL — 3 INTEGRATE |
| Viewpoint: Functionality View — DOM SYA SWA — 1 DESIGN | Perspective: Performance & Scalability — OPS DPL SYA SWA — 1 DESIGN |
| Viewpoint: Information View — DOM SWA — 1 DESIGN | Perspective: Availability & Recoverability — SYA SWA DPL OPS — 2 OPERATE |
| Viewpoint: Concurrency View — DEV SYA SWA — 2 DESIGN | Perspective: Reliability & Resilience — DEV SYA SWA — 2 DESIGN |
| Viewpoint: Development View — ASM VER SWA DEV — 3 IMPLEMENT | Perspective: Interoperability & Compatib. — DEV SWA — 3 DESIGN |
| Viewpoint: Deployment View — SYA SWA OPS DPL — 1 DEPLOY | Perspective: Compliance & Tracability — PRD TST SWA DEV — 3 IMPLEMENT |
| Viewpoint: Operations View — SYA SWA DPL OPS — 2 OPERATE | Perspective: Security & Safety — DEV SWA — 1 DESIGN |

## 3 Software **Implementation** Results *output / what* IMP

| | |
|---|---|
| Source Code: Application — REF DEV — 1 IMPLEMENT | Binary Code: Application — ASM — 1 BUILD |
| Source Code: Build Automation — DEV VER ASM — 2 BUILD | Source Code: Deployment Automation — DEV VER DPL — 3 DEPLOY |
| Source Code: Test Automation — DEV TST — 3 VERIFY | Source Code: Operation Automation — DEV OPS — 3 OPERATE |

### Notice: Artifacts vs. Aspects

The four **Artifact Sets** shown here just cluster the individual **Artifacts** and their contained **Aspects**. The **Artifacts** can be represented in an arbitrary graphical and/or textual form and be provided in an arbitrary format. The **Aspects** just structure an individual Artifact internally.

### Notice: Internal vs. External

In a Software Engineering project, additional **internal Artifacts** are created by the **Disciplines** in order to perform their work efficiently and effectively. The Artifacts shown here are the **external** ones which glue together the Disciplines and which are part of the delivery set.

## 4 Software **Documentation** Results *output / how* DOC

| | |
|---|---|
| User Guide: Usage Tutorial — UXP TRN DOC — 2 SPECIFY | Operation Guide: Configuration Reference — TRN DOC DPL DEV — 1 IMPLEMENT |
| User Guide: Functionality Reference — DOM TRN DOC — 1 SPECIFY | Operation Guide: Deployment Procedure — TRN DOC OPS DPL — 1 DEPLOY |
| User Guide: Release Information — PRD VER — 3 RELEASE | Operation Guide: Operation Procedures — TRN DOC DPL OPS — 2 OPERATE |

### Notice: Artifact Tagging

Each Artifact is tagged with the primarily and secondarily responsible **Disciplines**, the primary **Step** of the **Workflow** where the Artifact is developed, and the **Scalability Layer** (1 to 3, indicating more to lesser importance).

### Notice: Domain vs. Technology

The **Software Requirements Specification** and the **Software Documentation Results** primarily have a **domain-specific** focus. The **Software Architecture Specification** and the **Software Implementation Results** primary have a **technological** focus.

*artifacts & deliverables*

Intellectual Content: Version 1.00 (2021-10-10), Authored 2021 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.00 (2021-10-10), Copyright © 2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

# Modern Software Specification

TECHNISCHE UNIVERSITÄT MÜNCHEN

SpecBook

Hierarchical Object Model
Versatile Object to Markdown Mapping
Wiki-Style Object Linking

Strict Validation with Custom Schema
Object Model Diagram Visualisation
CLI, MCP, and API Interface

AST Exports for AI/LLMs
HTML Export for Developers
PDF Export for Customers

## SpecBook **SCHEMA Model** (YAML)



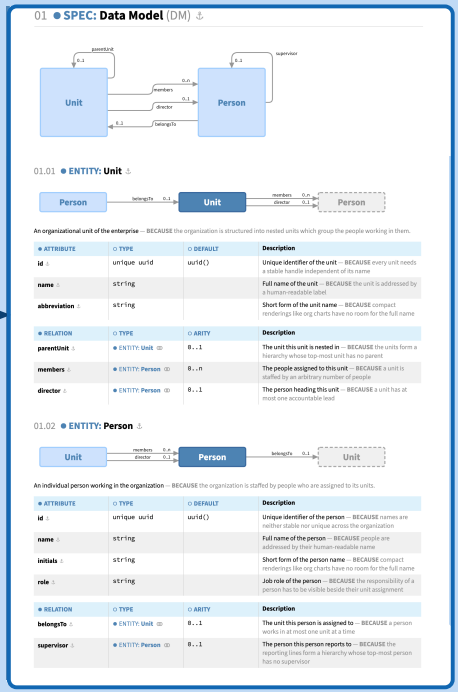
```yaml
- kind: META
  name: Title
  file: sample.md
  props:
      - name: TITLE
        desc: The title of the specification.
      - name: SUBTITLE
        desc: The subtitle of the specification.
        optional: true
- kind: SPEC
  name: Data Model
  file: sample.md
  id: DM
  desc: >
      The structure of named entities, with named attributes and directed relationships,
      of the data the solution manages, defining how information is organized and connected.
  diagram:
      type: graph
      nodes: "[[SPEC:DM.ENTITY:*]]"
      edges: "[[SPEC:DM.ENTITY:*.RELATION:*]]"
      config:
          size-node-height-scale:  8
          size-edge-port-gap:  60
          graph-node-degree-max:  4
          graph-channel-width-max: 300
          graph-channel-width-min: 300
  childs:
      - kind: ENTITY
        name: "[A-Z][a-zA-Z0-9]*"
        desc: >
            A named entity of the data model, carrying a Pascal-cased, 1-3 word long,
            per-artifact unique name, and described by a 1-sentence description ("what")
            plus a 1-sentence rationale ("why").
        diagram:
            type: hub
            center: self
            nodes: "[[SPEC:DM.ENTITY:*]]"
            edges: "[[SPEC:DM.ENTITY:*.RELATION:*]]"
            config:
              size-node-width-max: 440
        childs:
            - kind: ATTRIBUTE
              name: "[a-z][a-zA-Z0-9]*"
              desc: >
                  A named attribute of the entity, carrying a camel-cased, 1-3 word long,
                  per-entity unique identifier, and described by a 1-sentence description
                  ("what") plus a 1-sentence rationale ("why").
              props:
                  - name: TYPE
                    desc: >
                        The type of the entity attribute, one of `boolean`, `string`,
                        `integer`, `float`, `uuid`, `bytes`, `date`, `time`, `datetime`, or
                        `enum(XXX,YYY[,...])`, optionally prefixed with the qualifier
                        `unique` to indicate inclusion into the primary key of the entity,
                        and optionally suffixed with `?` to indicate an optional (nullable)
                        value.
                    value: /^(?:unique\s+)?(?:boolean|string|integer|float|uuid|datetime|enum)$/
                  - name: DEFAULT
                    desc: The default value of the entity attribute.
                    value: /.*/
                    optional: true
            - kind: RELATION
              name: "[a-z][a-zA-Z0-9]*"
              desc: >
                  A named directed relation of the entity, carrying a camel-cased, 1-3 word
                  long, per-entity unique identifier and targeting another entity, and
                  described by a 1-sentence description ("what") plus a 1-sentence rationale
                  ("why").
              optional: true
              props:
                  - name: TYPE
                    desc: Reference to the DATA-MODEL entity the directed relation targets.
                    value: "[[SPEC:DM.ENTITY:*]]"
                  - name: ARITY
                    desc: >
                        The arity (cardinality) of the relation at the target entity: `0..1`
                        or `?` for zero or one ("optional"), `1` for exactly one
                        ("mandatory"), `0..n` or `*` for zero or more, `1..n` or `+` for one
                        or more, or `{n}`/`{n,n}` for an exact number or range.
                    value: /^(?:1\?|\*|1\+|\{\d+(?,\d+)?\}|[1|0]\.\.1|0\.\.n|1\.\.n)$/
                    optional: true
```

## SpecBook **SPEC Model** (Markdown)



```
---
Created:  2026-08-23 00:15
Modified: 2026-08-23 13:03
---

META: Title
==========

-   TITLE:____Sample
-   SUBTITLE: Sample Specification

SPEC: Data Model (DM)
=====================

ENTITY: Unit
------------

An organizational unit of the enterprise, BECAUSE the organization is
structured into nested units which group the people working in them.

-   ATTRIBUTE: id; TYPE: `unique uuid`; DEFAULT: `uuid()`;
    Unique identifier of the unit, BECAUSE every unit needs a stable
    handle independent of its name.

-   ATTRIBUTE: name; TYPE: `string`;
    Full name of the unit, BECAUSE the unit is addressed by a
    human-readable label.

-   ATTRIBUTE: abbreviation; TYPE: `string`;
    Short form of the unit name, BECAUSE compact renderings like org
    charts have no room for the full name.

-   RELATION: parentUnit; TYPE: [[ENTITY:Unit]]; ARITY: `0..1`;
    The unit this unit is nested in, BECAUSE the units form a hierarchy
    whose top-most unit has no parent.

-   RELATION: members; TYPE: [[ENTITY:Person]]; ARITY: `0..n`;
    The people assigned to this unit, BECAUSE a unit is staffed by an
    arbitrary number of people.

-   RELATION: director; TYPE: [[ENTITY:Person]]; ARITY: `0..1`;
    The person heading this unit, BECAUSE a unit has at most one
    accountable lead.

ENTITY: Person
--------------

An individual person working in the organization, BECAUSE the
organization is staffed by people who are assigned to its units.

-   ATTRIBUTE: id; TYPE: `unique uuid`; DEFAULT: `uuid()`;
    Unique identifier of the person, BECAUSE names are neither stable
    nor unique across the organization.

-   ATTRIBUTE: name; TYPE: `string`;
    Full name of the person, BECAUSE people are addressed by their
    human-readable name.

-   ATTRIBUTE: initials; TYPE: `string`;
    Short form of the person name, BECAUSE compact renderings like org
    charts have no room for the full name.

-   ATTRIBUTE: role; TYPE: `string`;
    Job role of the person, BECAUSE the responsibility of a person has
    to be visible beside their unit assignment.

-   RELATION: belongsTo; TYPE: [[ENTITY:Unit]]; ARITY: `0..1`;
    The unit this person is assigned to, BECAUSE a person works in at
    most one unit at a time.

-   RELATION: supervisor; TYPE: [[ENTITY:Person]]; ARITY: `0..1`;
    The person this person reports to, BECAUSE the reporting lines form
    a hierarchy whose top-most person has no supervisor.
```

## SpecBook **SPEC Export** (AST, HTML, PDF)



01 ● **SPEC: Data Model** (DM) ⚓




01.01 ● **ENTITY: Unit** ⚓




An organizational unit of the enterprise — **BECAUSE** the organization is structured into nested units which group the people working in them.

| ● ATTRIBUTE | ○ TYPE | ○ DEFAULT | Description |
|---|---|---|---|
| **id** ⚓ | unique uuid | uuid() | Unique identifier of the unit — **BECAUSE** every unit needs a stable handle independent of its name |
| **name** ⚓ | string | | Full name of the unit — **BECAUSE** the unit is addressed by a human-readable label |
| **abbreviation** ⚓ | string | | Short form of the unit name — **BECAUSE** compact renderings like org charts have no room for the full name |

| ● RELATION | ○ TYPE | ○ ARITY | Description |
|---|---|---|---|
| **parentUnit** ⚓ | ● ENTITY: Unit 🆔 | 0..1 | The unit this unit is nested in — **BECAUSE** the units form a hierarchy whose top-most unit has no parent |
| **members** ⚓ | ● ENTITY: Person 🆔 | 0..n | The people assigned to this unit — **BECAUSE** a unit is staffed by an arbitrary number of people |
| **director** ⚓ | ● ENTITY: Person 🆔 | 0..1 | The person heading this unit — **BECAUSE** a unit has at most one accountable lead |

01.02 ● **ENTITY: Person** ⚓




An individual person working in the organization — **BECAUSE** the organization is staffed by people who are assigned to its units.

| ● ATTRIBUTE | ○ TYPE | ○ DEFAULT | Description |
|---|---|---|---|
| **id** ⚓ | unique uuid | uuid() | Unique identifier of the person — **BECAUSE** names are neither stable nor unique across the organization |
| **name** ⚓ | string | | Full name of the person — **BECAUSE** people are addressed by their human-readable name |
| **initials** ⚓ | string | | Short form of the person name — **BECAUSE** compact renderings like org charts have no room for the full name |
| **role** ⚓ | string | | Job role of the person — **BECAUSE** the responsibility of a person has to be visible beside their unit assignment |

| ● RELATION | ○ TYPE | ○ ARITY | Description |
|---|---|---|---|
| **belongsTo** ⚓ | ● ENTITY: Unit 🆔 | 0..1 | The unit this person is assigned to — **BECAUSE** a person works in at most one unit at a time |
| **supervisor** ⚓ | ● ENTITY: Person 🆔 | 0..1 | The person this person reports to — **BECAUSE** the reporting lines form a hierarchy whose top-most person has no supervisor |


Intellectual Content: Version 1.00 (2026-10-28), Authored 2026 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.00 (2026-10-28), Copyright © 2026 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.


EF 02.6

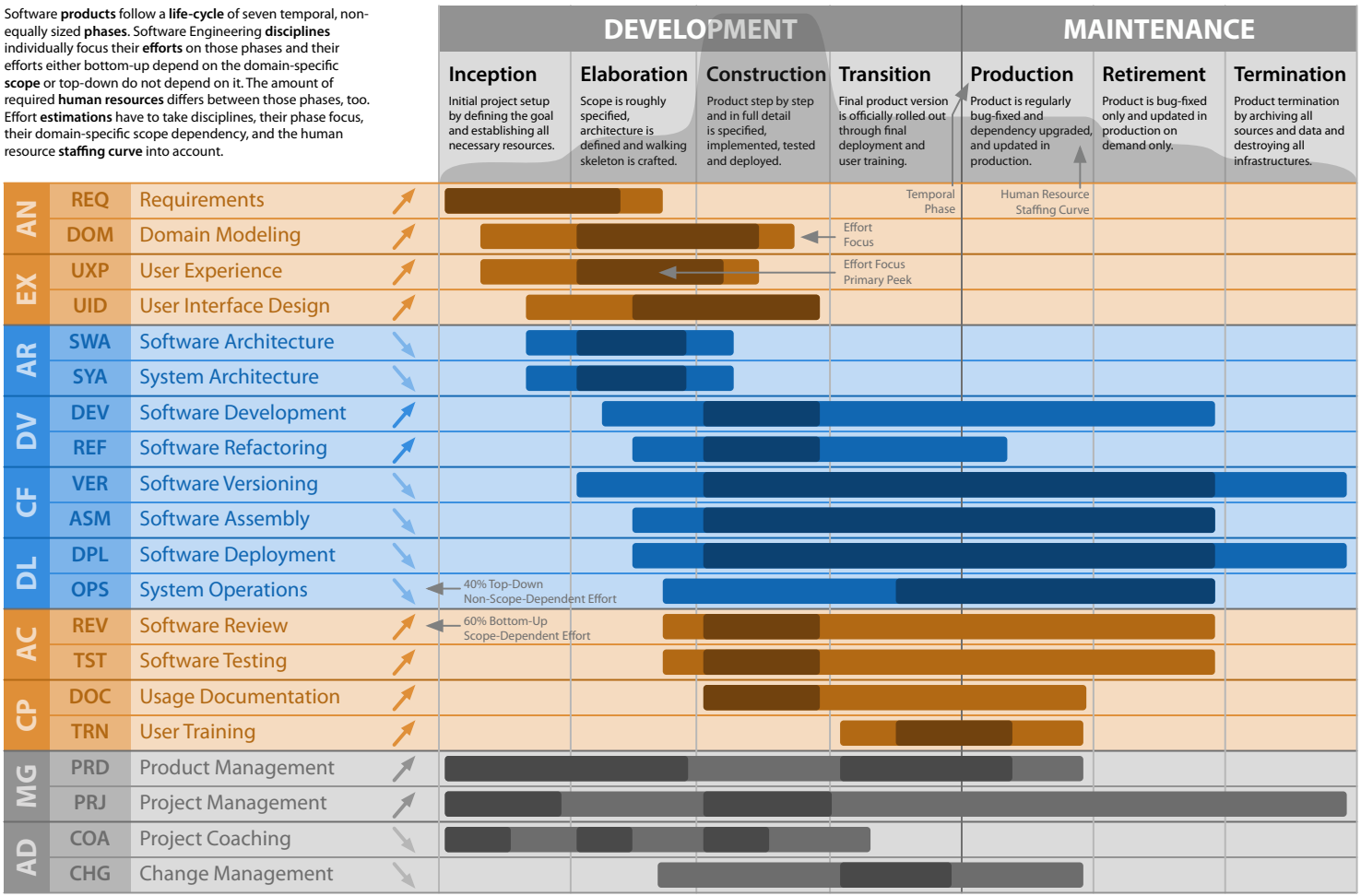
artifacts & modeling

ENGINEERING FUNDAMENTALS

# Software Engineering Efforts

TECHNISCHE UNIVERSITÄT MÜNCHEN

Software **products** follow a **life-cycle** of seven temporal, non-equally sized **phases**. Software Engineering **disciplines** individually focus their **efforts** on those phases and their efforts either bottom-up depend on the domain-specific **scope** or top-down do not depend on it. The amount of required **human resources** differs between those phases, too. Effort **estimations** have to take disciplines, their phase focus, their domain-specific scope dependency, and the human resource **staffing curve** into account.




EF 03.1

Intellectual Content: Version 1.0.4 (2022-08-11) Authored 2021-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.1 (2021-09-11) Copyright © 2021 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

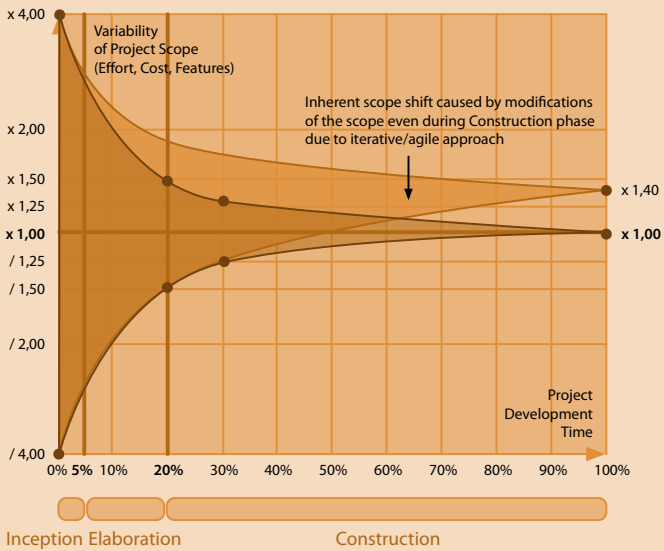
*phases & efforts*

# Uncertainty & Elaboration





## Cone of Uncertainty




The ***Cone of Uncertainty*** (*Steve McConnell*, 2006) tells how the variability of the project scope (measured in Effort, Cost or Features) in Software Development changes over time. Initially, it usually is within the range of +/- 400% of the final scope.

The early development phases Inception and Elaboration especially have to ensure that within the first 20% of the project, the variability is reduced noticeably to just +/- 50%. During the initial iterations of the Construction phase within the first 30% of the project, the variability usually can be further reduced to about +/- 25%.

For iterative/agile approaches, experience showed that during the Construction phase inherently the final scope further shifts by about + 40% due to the just step-by-step learned required details of the required solution. This especially has to be taken into account for estimations.

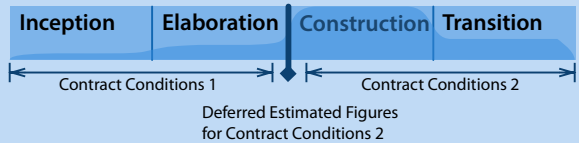
## Essential Elaboration Phase

### Walking Skeleton:

The ***Walking Skeleton*** (or *Technical Breakthrough*) is the design and implementation of the bare technical foundation of an application, still *without* any domain-specific functionalities. It is made during the Elaboration phase with the primary purpose to establish a stable integration of all technical aspects (libraries, frameworks, build procedures, etc) onto which the domain-specific functionalities later can be successively put onto.



### Agile Fixed-Price Contracts:




The ***Agile Fixed-Price*** is an agile variant of a fixed-price contract, *not* a fixed-price project with an agile development process.



There are two important inherent aspects:

First, the contract contains two types of conditions: one (usually *Time & Material* but fixed duration based) for the Inception and Elaboration phases in order to make experiences and to gather necessary figures, and one (usually Fixed-User-Story and/or Fixed-Price based) for the Construction and Transition phases based on deferred estimated figures, gathered in the Elaboration phase.

Second, the Fixed-Price aspect of the contract is actually based on an amount of User-Stories (resulting in costs by multiplying them with either an average hourly rate of an engineer or individual rates based on engineer job levels), which the customer can 1:1 *exchange* during the project for different deliverables.

The crux of an Agile Fixed-Price contract is: first, during the Inception and Elaboration phases the supplier can shrink the *Cone of Uncertainty* and this way its risks dramatically, and second, during the Construction and Transition phases the customer still remains flexible in scope.

Intellectual Content: Version 1.0.2 (2022-10-10), Authored 2021-2020 by Dr. Ralf S. Engelschall based on concepts from Steve McConnell.
Graphical Illustration: Version 1.0.1 (2022-10-10), Copyright © 2021-2020 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

# Effort Estimations

TECHNISCHE UNIVERSITÄT MÜNCHEN



## Estimation & Variability

### Three-Point Estimation and Estimation Variability Classes:

| | |
|---|---|
| $e = (b + 4 \times m + w) / 6$ | expected effort (weighted average) |
| $s = (w - b) / 6$ | standard deviation (effort variation) |

***b***: best-case (optimistic)
***m***: most-likely (realistic)
***w***: worst-case (pessimistic)

| | |
|---|---|
| Insane Variability: | +/- 10% |
| Very Good Variability: | +/- 15% |
| Good Variability: | +/- 20% |
| Acceptable Variability: | +/- 25% |

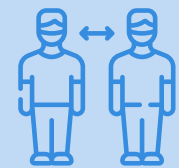


## Conversion & Normalization

**1. Ask Estimater:**
*"How many Person-Days do you need when you can focus on this task?"*

**2. Convert from Estimator to Performer:**
(see also CAP model, http://cap-model.com)

| Estimator \ Performer | Novice (0%) | Practitioner (10%) | Master (25%) | Expert (45%) | Guru (80%) |
|---|---|---|---|---|---|
| Novice | 1,00 | 0,90 | 0,75 | 0,55 | 0,20 |
| Practitioner | 1,11 | 1,00 | 0,83 | 0,61 | 0,22 |
| Master | 1,33 | 1,20 | 1,00 | 0,73 | 0,27 |
| Expert | 1,82 | 1,64 | 1,36 | 1,00 | 0,36 |
| Guru | 5,00 | 4,50 | 3,75 | 2,75 | 1,00 |

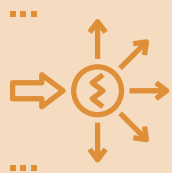


## Sizes & Variability

### Estimation Sizes and Estimation Variability:

| T-Shirt-Size (Logically) | XXS | XS | S | M | L | XL | XXL | XXXL |
|---|---|---|---|---|---|---|---|---|
| Fibonacci-Size (PD or SP) | **0,50** | **1** | **2** | **3** | **5** | **8** | **13** | **21** |
| Size Variability (-) | 0,25 | 0,25 | 0,50 | 0,50 | 1,00 | 1,50 | 2,50 | 4,00 |
| Size Variability (+) | 0,25 | 0,50 | 0,50 | 1,00 | 1,50 | 2,50 | 4,00 | 8,00 |

Notice: Estimations can be done in *Person-Days (PD)* or *Story-Points (SP)*. In both cases, keep in mind to use something like the *Fibonacci* numbers which increase in a non-linear fashion and express the increasing variability with the increasing total amount of estimated effort.



## Risk Mitigation & Upscaling

**3. Adjust for Reality:**
Estimator Optimism: +30%
Performer Meetings: +20%

**4. Adjust for Uncertainty:**

| Domain | Inception | Elaboration | Construction |
|---|---|---|---|
| unknown | 30% | 40% | 20% |
| partially known | 15% | 20% | 10% |
| fully known | 0% | 0% | 0% |

| Technology | Inception | Elaboration | Construction |
|---|---|---|---|
| unknown | 20% | 60% | 10% |
| partially known | 10% | 30% | 5% |
| fully known | 0% | 0% | 0% |

| Process | Inception | Elaboration | Construction |
|---|---|---|---|
| unknown | 60% | 40% | 10% |
| partially known | 30% | 20% | 5% |
| fully known | 0% | 0% | 0% |

| People | Inception | Elaboration | Construction |
|---|---|---|---|
| unknown | 60% | 40% | 0% |
| partially known | 30% | 20% | 0% |
| fully known | 0% | 0% | 0% |



Intellectual Content: Version 1.0.1 (2022-10-10), Authored 2021-2022 by Dr. Ralf S. Engelschall based on concepts from Steve McConnell.
Graphical Illustration: Version 1.0.1 (2022-10-10), Copyright © 2021-2022 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

# Agentic Software Engineering

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 05.1

## SW Software 1

The collection of programs (instructions) and their associated data that make a computer perform tasks. In contrast to hardware, it is intangible and can be changed, copied, and distributed without altering the physical machine.

## SA Software Agent 2

**Software** that acts autonomously on behalf of a user or another program to achieve delegated goals. Exhibits reactivity (responds to environment), proactivity (goal-directed initiative), and social ability (interacts with other agents/users). Perceives its environment via sensors/inputs (APIs, files, messages, events) and acts via actuators/outputs (commands, calls, messages).

## EN Engineering 3

The systematic application of scientific and technological knowledge, principles, approaches, practices, methods, and experiences to design, build, operate, and maintain solutions, up to complex production-grade ones, through a traceable process.

## SE Software Engineering 4

The application of **Engineering** to the development, operation, and maintenance of **Software**.

## AI Artificial Intelligence 5

The field of Computer Science focusing on building systems that perform tasks normally requiring human intelligence: perception, reasoning, learning, planning, decision-making, and language understanding. It spans Symbolic AI (rules, logic, search) and Machine Learning (including Deep Learning and Generative AI such as LLMs and diffusion models), ranging from narrow, task-specific AI to Artificial General Intelligence (AGI).

## AH Agent Harness 6

The **Software** that surrounds an AI model and turns it from a passive, predicting model into an acting **Software Agent**. It implements a control loop (e.g. ReAct or Plan-and-Execute) that sends context to the model, executes the requested actions, and feeds results back, and it supplies the tools (sensors and actuators), context management, and guardrails. The model decides what to do; the harness determines what it can do and how.

## AG AI Agent 7

A **Software Agent** formed by an AI model (typically an Large Language Model, LLM) as its reasoning core, operated by an **Agent Harness**.

## AA Agentic AI 8

The paradigm of building AI systems out of one or more **AI Agents** that perceive, plan, decide, act, and adapt to pursue high-level, long-horizon goals with a high degree of autonomy, typically coordinated through task decomposition, orchestration, and shared context. Unlike purely generative (prompt-response) AI, its systems take actions in the world (call APIs, run code, edit files, browse the web).

## AS Agentic Software Engineering 9

**Software Engineering** following the **Agentic AI** paradigm, in which **AI Agents** perform multi-step tasks across the software development lifecycle, within a traceable process. Goes beyond inline code completion and chat assistants. Engineers shift toward goal specification, oversight, review, and governance, with human-in-the-loop gates.

*concepts & relationships*

Intellectual Content: Version 1.0.0 (2026-10-28), Authored 2026 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.0.0 (2026-10-28), Copyright © 2026 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

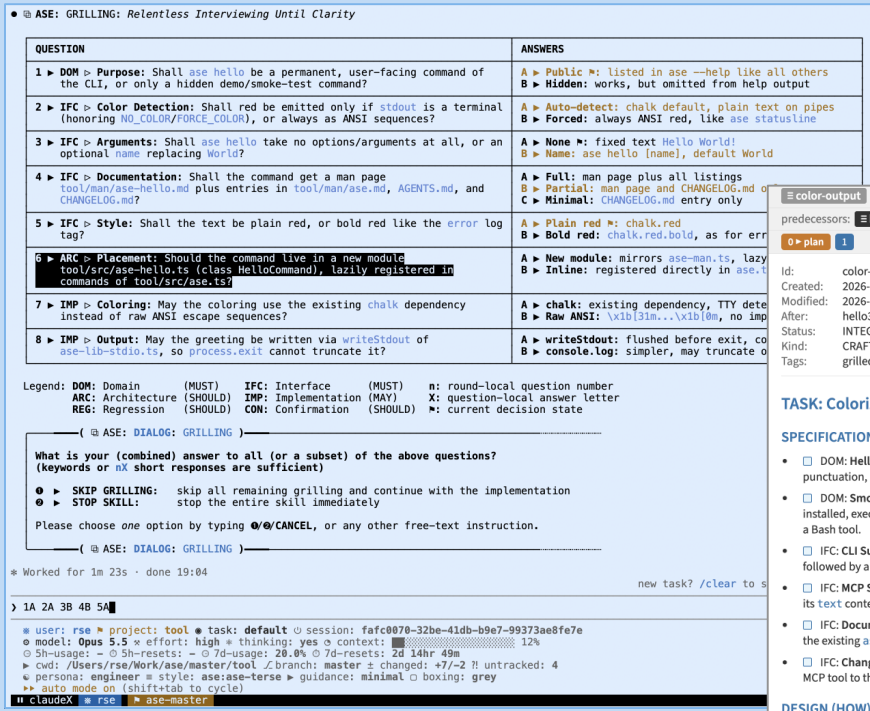
# Agentic Modus Operandi

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 05.2

| | | | | | |
|---|---|---|---|---|---|
| Actor | HUMAN ⟶ AI AGENT | | | | |
| Modus Operandi | ⟵ ⟵ ⟵ ⟵ ⟶ | | | | |
| Level | 0 manually | 1 assisted | 2 supervised | 3 delegated | 4 autonomous |
| View |  white |  grey / light |  grey |  grey / dark |  black |
| Objective | none | step | task | workflow | lifecycle |
| Think | self | assistant | co-worker | offshoring | 3rd-party |
| Solution | life-critical | safety-related | mission-critical | business-supporting | non-critical |
| Productivity | costs ⟶ efficiency | | | | |
| Knowledge | do-it-yourself ⟶ reviewing | | | | |
| Traceability | do-it-yourself ⟶ evidence | | | | |
| Briefing | do-it-yourself ⟶ specification | | | | |

Intellectual Content: Version 1.0.0 (2026-10-28), Authored 2026 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.0 (2026-10-28), Copyright © 2026 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved
Unauthorized Reproduction Prohibited, Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.



● ASE: GRILLING: *Relentless Interviewing Until Clarity*

| QUESTION | ANSWERS |
|---|---|
| 1 ▶ **DOM** ▷ **Purpose:** Shall `ase hello` be a permanent, user-facing command of the CLI, or only a hidden demo/smoke-test command? | A ▶ **Public** ⚑: listed in `ase --help` like all others<br>B ▶ **Hidden:** works, but omitted from help output |
| 2 ▶ **IFC** ▷ **Color Detection:** Shall red be emitted only if `stdout` is a terminal (honoring `NO_COLOR`/`FORCE_COLOR`), or always as ANSI sequences? | A ▶ **Auto-detect:** chalk default, plain text on pipes<br>B ▶ **Forced:** always ANSI red, like `ase statusline` |
| 3 ▶ **IFC** ▷ **Arguments:** Shall `ase hello` take no options/arguments at all, or an optional `name` replacing `World`? | A ▶ **None** ⚑: fixed text `Hello World!`<br>B ▶ **Name:** `ase hello [name]`, default `World` |
| 4 ▶ **IFC** ▷ **Documentation:** Shall the command get a man page `tool/man/ase-hello.md` plus entries in `tool/man/ase.md`, `AGENTS.md`, and `CHANGELOG.md`? | A ▶ **Full:** man page plus all listings<br>B ▶ **Partial:** man page and `CHANGELOG.md` o<br>C ▶ **Minimal:** `CHANGELOG.md` entry only |
| 5 ▶ **IFC** ▷ **Style:** Shall the text be plain red, or bold red like the `error` log tag? | A ▶ **Plain red** ⚑: `chalk.red`<br>B ▶ **Bold red:** `chalk.red.bold`, as for err |
| 6 ▶ **ARC** ▷ **Placement:** Should the command live in a new module `tool/src/ase-hello.ts` (class `HelloCommand`), lazily registered in `commands of tool/src/ase.ts`? | A ▶ **New module:** mirrors `ase-man.ts`, lazy<br>B ▶ **Inline:** registered directly in `ase.t` |
| 7 ▶ **IMP** ▷ **Coloring:** May the coloring use the existing `chalk` dependency instead of raw ANSI escape sequences? | A ▶ **chalk:** existing dependency, TTY dete<br>B ▶ **Raw ANSI:** `\x1b[31m...\x1b[0m`, no imp |
| 8 ▶ **IMP** ▷ **Output:** May the greeting be written via `writeStdout` of `ase-lib-stdio.ts`, so `process.exit` cannot truncate it? | A ▶ **writeStdout:** flushed before exit, co<br>B ▶ **console.log:** simpler, may truncate o |

Legend: **DOM:** Domain (MUST) **IFC:** Interface (MUST) **n:** round-local question number
**ARC:** Architecture (SHOULD) **IMP:** Implementation (MAY) **X:** question-local answer letter
**REG:** Regression (SHOULD) **CON:** Confirmation (SHOULD) **⚑:** current decision state

——( ASE: DIALOG: GRILLING )——

**What is your (combined) answer to all (or a subset) of the above questions?**
**(keywords or nX short responses are sufficient)**

❶ ▶ **SKIP GRILLING:** skip all remaining grilling and continue with the implementation
❷ ▶ **STOP SKILL:** stop the entire skill immediately

Please choose *one* option by typing **❶/❷/CANCEL**, or any other free-text instruction.

——( ASE: DIALOG: GRILLING )——

✻ Worked for 1m 23s · done 19:04

new task? `/clear` to s

❯ 1A 2A 3B 4B 5A

✻ user: rse ⚑ project: tool ◉ task: **default** ⏻ session: **fafc0070-32be-41db-b9e7-99373ae8fe7e**
⚙ model: **Opus 5.5** ✲ effort: **high** ✳ thinking: **yes** ◔ context: ▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓ 12%
⏲ 5h-usage: – ⏲ 5h-resets: – ⏲ 7d-usage: **20.0%** ⏲ 7d-resets: **2d 14hr 49m**
▶ cwd: **/Users/rse/Work/ase/master/tool** ⎇ branch: **master** ± changed: **+7/−2** ?! untracked: **4**
☯ persona: **engineer** ≡ style: **ase:ase-terse** ▶ guidance: **minimal** ▢ boxing: **grey**
▸▸ auto mode on (shift+tab to cycle)
claudeX ✻ rse ⚑ ase-master

**Brief Engineering:** The systematic specification of a delegated assignment for an AI agent: goals, scope, constraints, acceptance criteria, and safety boundaries. It defines *what* must be achieved and how success is verified, so an agent can work autonomously across long-horizon tasks and deliver traceable evidence.

color-output Colorize CLI Output on Terminals — 5 - Done ▷ INTEGRATED

predecessors: hello3 — successors: —

0 ▸ plan 1

Id: color-output
Created: 2026-08-03 00:21
Modified: 2026-09-13 19:49
After: hello3
Status: INTEGRATED
Kind: CRAFTING
Tags: grilled:true

## TASK: Colorize CLI Output on Terminals

### SPECIFICATION (WHAT)

- ☐ DOM: **Hello Command**: Add a new top-level CLI command `ase hello` which prints exactly `Hello World` (no punctuation, no version suffix) to the terminal in *red* color, plus a companion `ase_hello` MCP tool.
- ☐ DOM: **Smoke Test**: Provides a minimal, self-contained *smoke-test* command for verifying that the `ase` binary is installed, executable, and able to emit colored terminal output, and that the MCP bridge is reachable from a skill without a Bash tool.
- ☐ IFC: **CLI Surface**: `ase hello` takes *no* arguments and *no* options beyond the standard `--help`, prints `Hello World` followed by a trailing newline, colored *red* on a TTY and *plain* when piped into a file.
- ☐ IFC: **MCP Surface**: The `ase_hello` MCP tool is *argument-less* and returns the plain, un-colored text `Hello World` as its `text` content.
- ☐ IFC: **Documentation**: Add an `ase hello` bullet point to the `COMMANDS` section of `docs/usage-tool.md`, in the style of the existing `ase compat` entry and introduced by its own lead-in sentence.
- ☐ IFC: **ChangeLog**: Add a `FEATURE [code]` bullet point announcing the new `ase hello` command and `ase_hello` MCP tool to the topmost section of `CHANGELOG.md`, matching its existing style and category conventions.

### DESIGN (HOW)

- ☐ ARC: **Single Module**: Create `tool/src/ase-hello.ts` as the *sole* new module, carrying both the CLI and the MCP surface, so the greeting text lives in *one* shared constant `GREETING = "Hello World"` and cannot drift between the two surfaces.
- ☐ ARC: **Command Class**: Export a default `HelloCommand` class with a `register (program: Command): void` method mirroring `CompatCommand` in `tool/src/ase-compat.ts`: it registers `program.command("hello")` with a short description and an async action, and takes *no* `Log` dependency.
- ☐ ARC: **MCP Class**: Export a named `HelloMCP` class with a `register (mcp: McpServer): void` method mirroring `CompatMCP`, registering the tool `ase_hello` via `mcp.registerTool(...)` with an *empty* `inputSchema`, a title, and a description.

←/→/⇤/⇥ switches tab · ↑/↓/⇞/⇟ scrolls · e edits · E edits by prompt · R requests by prompt · T transitions · A assigns · D deletes · ↵/ESC closes

**Prompt Engineering:** The craft of formulating individual inputs (prompts) to an AI model so that a single request reliably produces the desired output. It focuses on phrasing, structure, examples, and instructions within one prompt–response exchange.

**Context Engineering:** The systematic design and management of everything an AI model sees while it works: instructions, retrieved knowledge, tool definitions and results, memory, and conversation history. It decides what goes into the limited context window at each step of an agent's control loop, usually handled by the Agent Harness.

Intellectual Content: Version 1.0.0 (2026-10-28). Authored 2026 by Dr. Ralf S. Engelschall (http://engelschall.com). All Rights Reserved.
Graphical Illustration: Version 1.0.0 (2026-10-28). Copyright © 2026 Dr. Ralf S. Engelschall. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

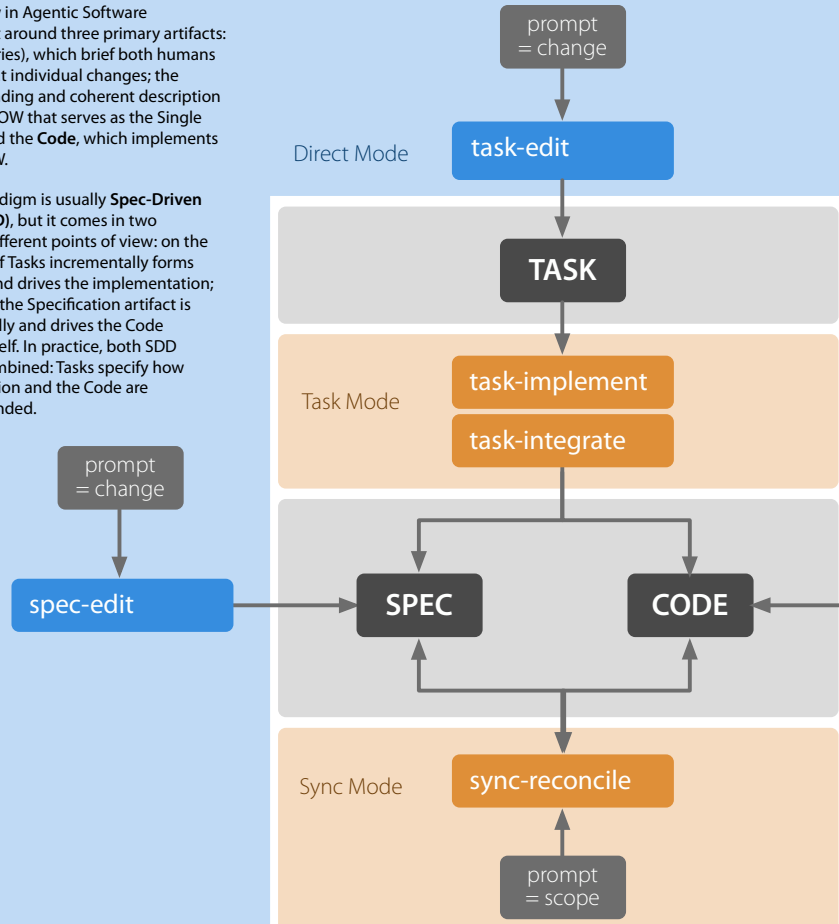
# Agentic Workflow

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 05.4

The usual workflow in Agentic Software Engineering is built around three primary artifacts: **Tasks** (aka user stories), which brief both humans and AI agents about individual changes; the **Specification**, a binding and coherent description of the WHAT and HOW that serves as the Single Source of Truth; and the **Code**, which implements the WHAT and HOW.

The dominant paradigm is usually **Spec-Driven Development (SDD)**, but it comes in two approaches with different points of view: on the one hand, the set of Tasks incrementally forms the Specification and drives the implementation; on the other hand, the Specification artifact is crafted incrementally and drives the Code implementation itself. In practice, both SDD approaches are combined: Tasks specify how both the Specification and the Code are incrementally extended.




| | Ad-Hoc Mode (Standard) | Direct Mode | Plan Mode (Standard) | Task Mode | Sync Mode |
|---|---|---|---|---|---|
| Direct Instruction | X | X | | | |
| Intention Grilling | | X | | X | |
| Tenet-Driven | | X | X | X | X |
| Deferred Execution | | | X | X | X |
| Plan Refinement | | | X | X | X |
| Parallel Plans | | | | X | |
| Cross-Session Plans | | | | X | X |
| Use-Case Steering | | X | | X | |
| Solution Approach Choices | | X | | X | |

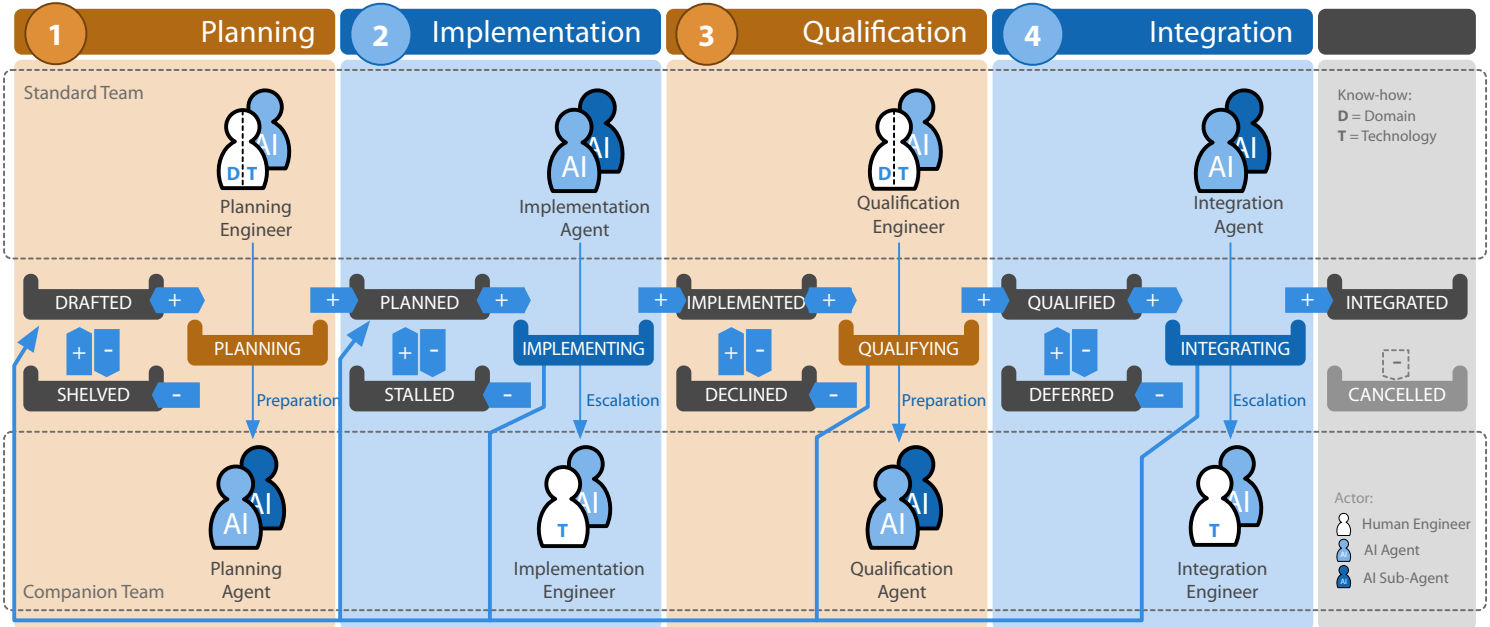
Intellectual Content: Version 1.00 (2026-10-28), Authored 2026 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.00 (2026-10-28), Copyright © 2026 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

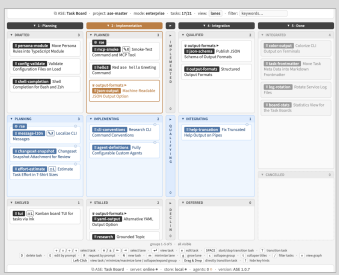
*concepts & relationships*

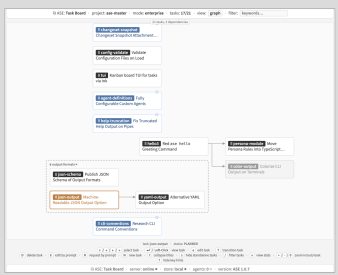
ENGINEERING FUNDAMENTALS

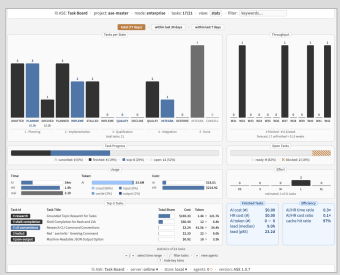
# Agentic Development Pipeline

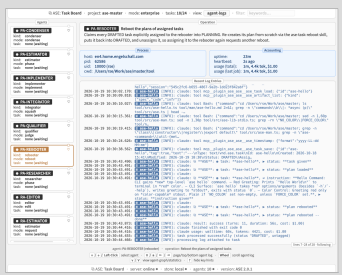
TECHNISCHE UNIVERSITÄT MÜNCHEN









EF 05.5

Intellectual Content: Version 1.00 (2026-10-28), Authored 2026 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.00 (2026-10-28), Copyright © 2026 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized reproduction in any form is licensed to technische Universität München exclusively for non-commercial use in computer science lecture contexts only.

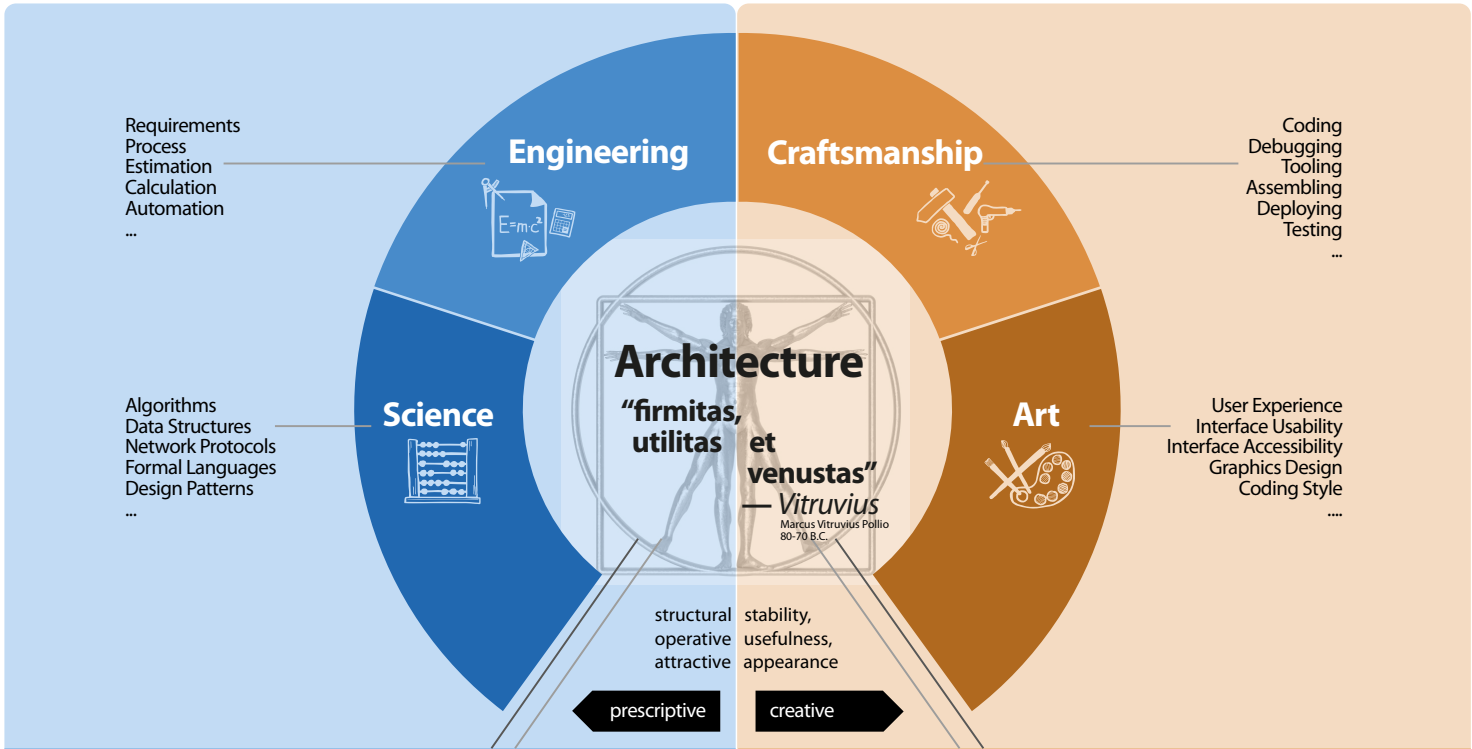
tasks & states

ARCHITECTURE FUNDAMENTALS

# Architecture Stargate

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 01.1




**Structural Scientific Definition of Architecture:**
Architecture of a system is the set of fundamental **concepts** and **properties** of the system in its environment, embodied in its **elements**[, **interfaces**], **relationships**[ and **behaviours**] and the principles of its **design** and **evolution**.
(based on industry standard *ISO/IEC 42010*)

**Holistic Artistic Definition of Architecture:**
Great architecture is achieved **harmony** and **accord** of all **parts**, where you no longer can add, modify or remove anything without impairing the **whole**.
(based on a quote by *Leon Battista Alberti* on beauty)

Intellectual Content: Version 1.0.8 (2020-11-28), Authored 2010-2020 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.7 (2020-07-29), Copyright © 2011-2020 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved
Unauthorized Reproduction Prohibited. **Licensed to Technische Universität München TUM for reproduction in Computer Science lecture contexts only.**

ARCHITECTURE FUNDAMENTALS

# Adequacy and Beauty

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 01.2

Intellectual Content: Version 1.0.2 (2022-10-09), Authored 2021-2022 by Dr. Raff S. Engelschall, based on pre-work of Stefan Bratzch
Graphical Illustration: Version 1.0.2 (2022-10-09), Copyright © 2021-2022 Dr. Raff S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture courses only.

## ADEQUACY

1. **Measure**, how **suitable**
2. something,
3. in its environment,
1. fulfills
4. its imposed functional and qualitative requirements (fitness for purpose),
5. from the **individual** and **subjective** perspective of a person.

Beauty does imply Adequacy

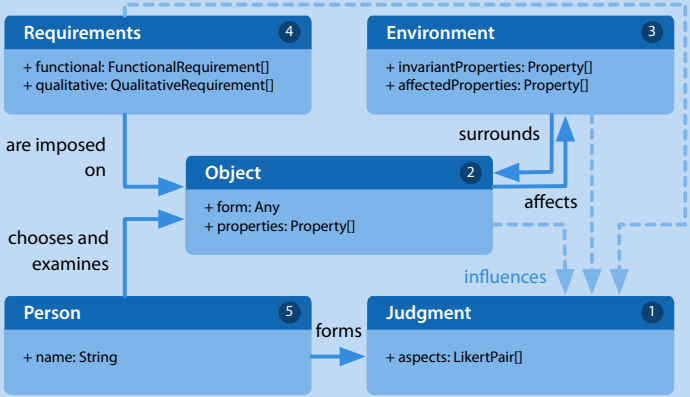
No Beauty does *not* imply No Adequacy

No Adequacy does imply No Beauty

Adequacy does *not* imply Beauty

## BEAUTY

**Perception**, how **simple, elegant, natural, securing and evident** (SENSE) something is, from the **evolutionary** and **cultural** perspective of a person.




**Simple** S — free from *unnecessary* complexity, easy *enough* to comprehend.

**Elegant** E — broadly applicable, allows combinatorial use-cases.

**Natural** N — consistent, symmetric, free of anomalies.

**Securing** S — allows *feeling* safe by captured experiences or omnipresence.

**Evident** E — obvious, clear and traceable representation.

**Adequacy** is a **relative** and highly **subjective measure** that defies scientific objectification. Nevertheless, it is an essential concept, as the **three influencing aspects** allow persons in practice to at least better **structure** their **perception and judgment** and **measure** the **differences** over time.

**Beauty** is a **relative** and highly **subjective perception** that defies scientific objectification. Nevertheless, it is an essential concept, as the **SENSE aspects** *seem* to be universal across persons and allow them to **communicate** their **perception and judgment** and **observe** the **differences** over time.

ARCHITECTURE FUNDAMENTALS

# King Discipline Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 01.3

Intellectual Content: Version 1.00 (2022-10-09), Authored 2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.00 (2022-10-09), Copyright © 2022 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

## Domain

Many specific Domains

**N**

**Domain**
**Analytics**
**Business**

domain-specific
usage-oriented
specific

## Architecture

One (or few) generic Architectures

**1**

**Architecture**
**Design**
**Construction**

conceptual
constructional
generic

## Technology

Many specific Technologies

**M**

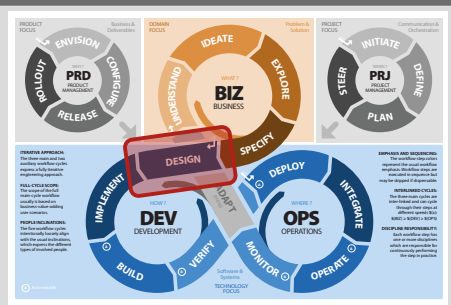
**Technology**
**Development**
**Realisation**

technological
programmatic
specific

## Software Engineering Disciplines




## Software Engineering Workflow



# Manifesto for IT Architecture

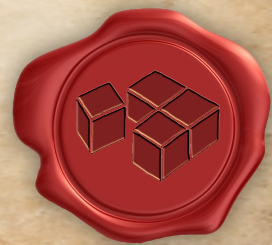
## Continuously Raising the Bar

*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.

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



Intellectual Content: Version 1.02 (2013-11-02), Authored 2013 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.11 (2018-11-26), Copyright © 2013-2018 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved
Unauthorized Reproduction Prohibited

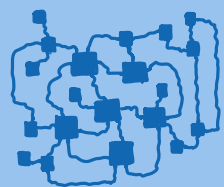
ARCHITECTURE FUNDAMENTALS

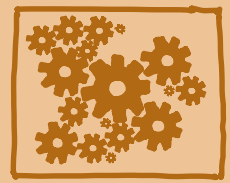
# Complex vs. Complicated

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 01.5

| | complex | complicated |
|---|---|---|
| FOCUS | refers to the **extrinsic** and **higher**- or **macro**-level difficulty of a system, | refers to the **intrinsic** and **lower**- or **micro**-level difficulty of a system, |
| RATIONALE | because the system involves many different and **connected** parts | because the system involves many different and **difficult** aspects |
| CHALLENGE | which take time to **comprehend** and **master** in **total**, | which take time to **understand** and **learn** in **detail**, |
| INSIGHT | and which nevertheless are **easy** to explain. | and which usually are **hard** to explain. |





| NOTICE | NOTICE |
|---|---|
| **Simple** (non-complicated) systems can be **complex**. | **Clear** (non-complex) systems can be **complicated**. |

| RECOGNIZE | RECOGNIZE |
|---|---|
| **Architecture** primarily has to master the **complex** aspects of a system. | **Development** primarily has to master the **complicated** aspects of a system. |

Intellectual Content: Version 1.00 (2019-10-12), Authored 2019 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.00 (2019-10-12), Copyright © 2019 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

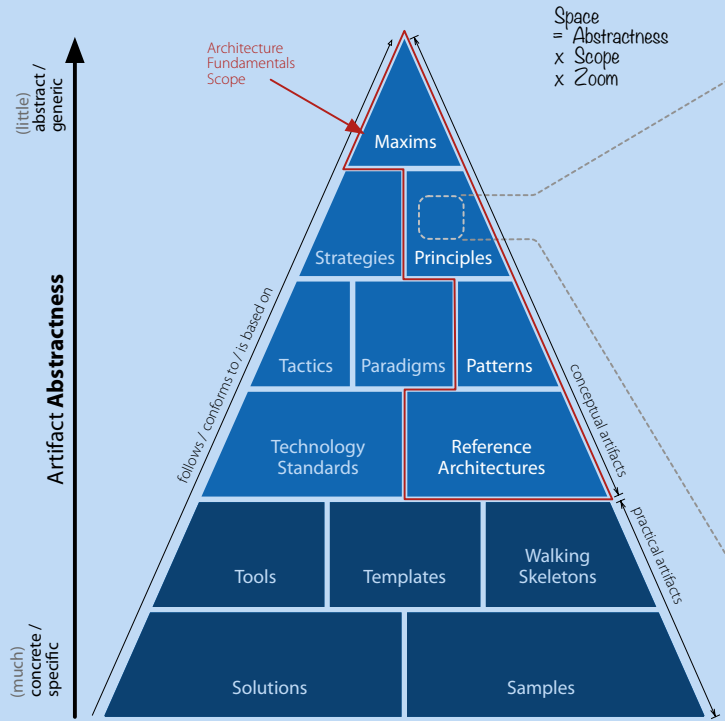
ARCHITECTURE FUNDAMENTALS

# Architecture Space

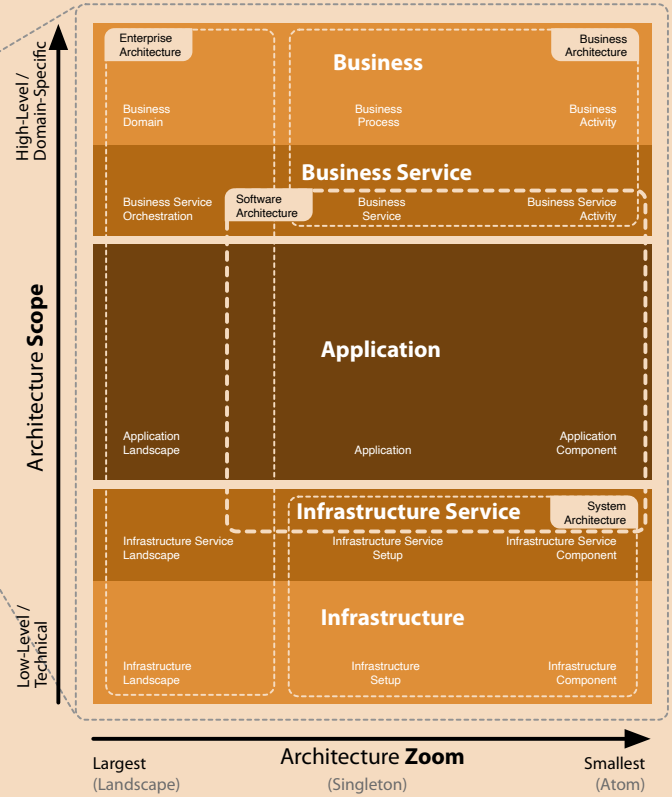
TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 02.1

## Artifact Abstractness




## Architecture Scope & Zoom

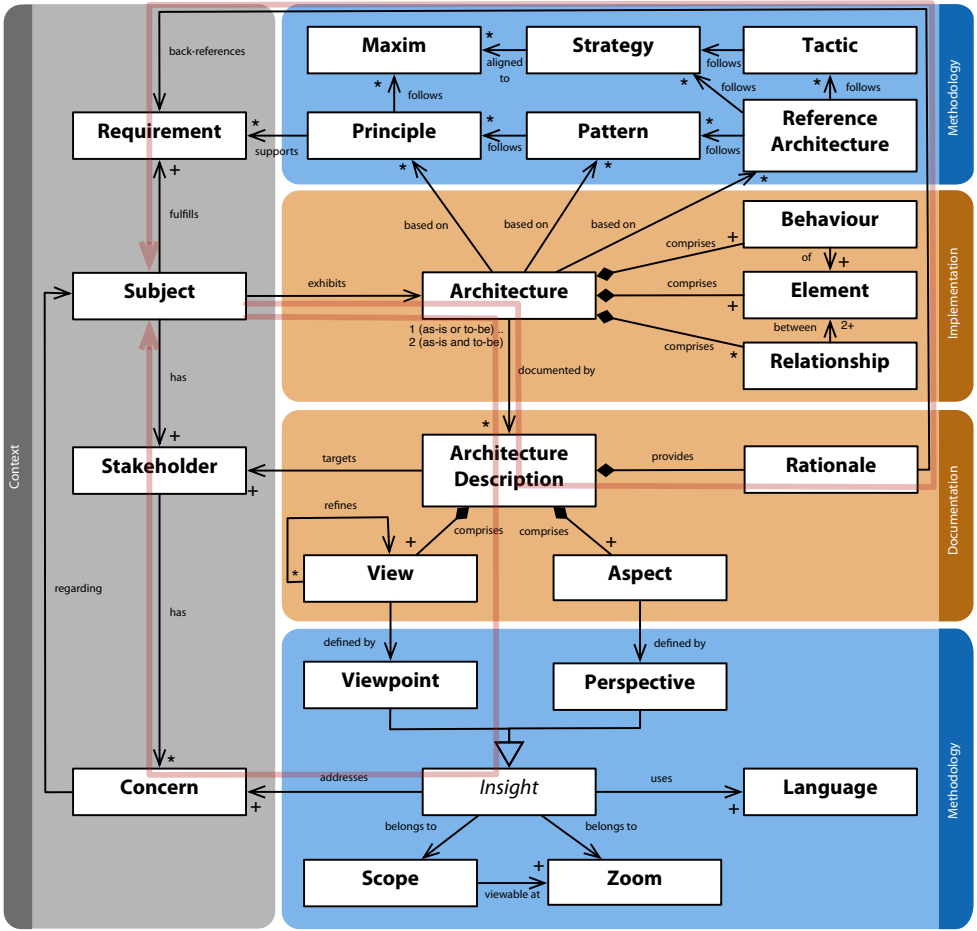



Intellectual Content: Version 1.0.9 (2020-06-23) Authored 2011-2020 by Dr. Ralf S. Engelschall and Marc Mühlfriedel, based on prior work of msg Applied Technology Research
Graphical Illustration: Version 1.1.2 (2019-09-15) Copyright © 2011-2019 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Architecture Ontology

TECHNISCHE UNIVERSITÄT MÜNCHEN




**Requirement:** A functional or non-functional demand or imposed obligation on the **Subject**.

**Architecture:** Inherent static and dynamic structure of a **Subject** which comprise **Elements**, the visible **Behaviour** of **Elements** and **Relationships** between **Elements**.

**Maxim:** Fundamental, generally valid set of values and rules to guide the architecture discipline (think: law).

**Subject:** Any type of business process, business service, software application, infrastructure service or infrastructure setup.

**Element:** Fundamental part from which a **Subject** can be considered to be constructed.

**Strategy:** Long-term situation-independent plan of approaches to achieve a particular goal (think: war). Aligned and not in conflict with any **Maxims**.

**Stakeholder:** Person, group or entity with an interest in or **Concerns** about the **Architecture**.

**Behaviour:** Run-time characteristic of **Architecture Elements** of the **Subject**.

**Tactic:** Short-term situation-dependent plan of actions to achieve a particular goal (think: battle). Following and supporting a **Strategy**.

**Concern:** Requirement, objective, intention, or aspiration a **Stakeholder** has on an **Architecture**.

**Relationship:** Static or dynamic relationship between **Elements** of an **Architecture** of the **Subject**.

**Principle:** Fundamental truth, rule, tenet or policy an **Architecture** follows.

**Architecture Description:** Set of artifacts that document an **Architecture** in a way **Stakeholders** can understand and ensures their **Concerns** are met.

**Rationale:** Fundamental reasons for a particular chosen **Architecture**, usually strongly based on non-functional **Requirements**.

**Pattern:** Proven recurring theme, structure, approach or behavior an **Architecture** and its **Elements** can follow.

**Viewpoint:** Collection of templates and conventions for constructing one type of **View**. It addresses **Concerns** and contains guidelines for constructing a View.

**View:** Representation of one or more structural aspects of an **Architecture** that illustrates how the **Architecture** addresses one or more **Concerns**.

**Reference Architecture:** Reusable proven Architecture template, **based** on a set of Patterns and following one or more **Tactics** and **Strategies**.

**Perspective:** Collection of decisions, guidelines and rules that ensure a **Subject** exhibits a set of non-functional **Requirements**, considered across a number of **Views**.

**Aspect:** Representation of one or more non-functional aspects of an **Architecture** that illustrate how the **Architecture** addresses one or more **Concerns**.

**Scope:** Primary area of the **Architecture** space an **Insight** addresses: either Business, Business Service, Application, Infrastructure Service or Infrastructure.

**Insight:** Superordinate abstract concept to address particular **Concerns** through **Viewpoints** and **Perspectives**.

**Language:** Formal, semi-formal or even prose language to describe an Insight, so it optimally addresses the **Concerns** of **Stakeholders**.

**Zoom:** Level of view distance to the **Architecture** an **Insight** has. Also known as the **Insight** detail level, ranging from smallest atoms, over singletons to the largest landscape.

AF 02.2

Intellectual Content: Version 1.1.8 (2020-08-13). Authored 2010-2020 by Dr. Ralf S. Engelschall, inspired by ISO/IEC/IEEE 42010:2011, IEEE 1471:2000 and The Open Group TOGAF 9. Graphical Illustration: Version 1.1.1 (2019-09-21). Copyright © 2011-2019 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved. Unauthorized reproduction prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Requirements Basics

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 03.1

## Requirements Specification

A binding document that specifies the requirements for a solution, by focusing on the WHAT and WHY of the solution — and *not* giving instructions for the HOW.

The documented set of requirements has to be: correct, unambiguous, complete, consistent, ranked, verifiable, modifiable, and traceable.



## Requirement Classes

### FR Functional (Shall Do)

A condition or capability that a solution must have to provide its service in terms of its behaviour and information. Think: Functionality.



### NFR Non-Functional (Shall Be)

A condition, property or quality that a solution must have to satisfy a contract, standard, or other formally imposed obligation. Think: Constraints and "*-ilities".



## Requirement Interdependencies

### POS Positive (Backing)

One requirement supports the other (e.g. for NFRs: Maintainability and Comprehensibility usually support Adaptability, Portability, Modifiability, etc., and Scalability usually supports Availability, etc.)



### NEG Negative (Trade-Off)

One requirement interferes with the other (e.g. for NFRs: Security usually interferes with Efficiency, Usability, Performance, etc., and Orthogonality can interfere with Usability)



## Requirement Characteristics

### S Specific

The requirement is precise, unambiguous, and clear on what should be done.



### M Measurable

The requirement can be verified when it has been achieved by use of a particular test.



### A Achievable

The requirement is achievable given existing circumstances and feasible and viable solutions.



### R Relevant

The requirement is relevant to the goals of the context.



### T Time-Bound

The requirement can be achieved within a reasonable time frame.



## Requirement Life-Time

### E Enduring

The requirement lasts forever, as it is derived from core activities and organisational structures.



### V Volatile

The requirement can be temporary, as it might change over time.



## Requirement Expression

[<req-id>] <req-name>:
<subject/actor>
**SHALL**
<result/action/condition>
**BECAUSE**
<rationale>



Intellectual Content: Version 1.0.2 (2022-10-28), Authored 2018-2022 by Dr. Ralf S. Engelschall, based on ideas from BABOK 2.0 2009, IEEE Std 610.1 1990 and IEEE Std 830-1998
Graphical Illustration: Version 1.0.2 (2022-10-09), Copyright © 2018-2022 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to TECHNISCHE UNIVERSITÄT MÜNCHEN for reproduction in Computer Science lecture courses only.

ARCHITECTURE FUNDAMENTALS

# Non-Functional Requirements

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 03.2

## Compliance

- **CMP Compliance** — Ability to meet rules and standards
- **CRT Certification** — Ability to confirm certain characteristics
- **LCN Licensing** — Ability to permit to own and use something
- **PRC Pricing** — Ability to have reasonable price and permit charging for a product

## Operation

- **OPR Operability** — Ability to be reasonably operated
- **SPP Supportability** — Ability to be reasonably supported
- **MNT Maintainability** — Ability to cope with changing environments and requirements
- **TST Testability** — Ability to be completely and repeatably tested
- **TRC Traceability** — Ability to track the path something takes
- **MSR Measurability** — Ability to measure characteristics according to defined metrics

## Usability

- **USB Usability** — Ability for ease of use, user-friendliness, accessibility, convenience, intuitiveness
- **CPY Comprehensability** — Ability to be easily understood
- **ACC Accessibility** — Ability to be used by people with disabilities.

• • •

## Correctness

- **PRD Predictability** — Ability to predict state and behaviour under run-time
- **FDL Fidelity** — Ability to reproduce state and behaviour of the real world
- **RLV Relevance** — Ability to serve as a means to a given purpose
- **PRN Precision** — Ability to be exact and accurate in operation
- **CRS Correctness** — Ability to be algorithmically correct with respect to the specification
- **PRV Provability** — Ability to mathematically prove algorithmical correctness

## Protection

- **SFT Safety** — Ability to protect against undeliberate failures, errors and accidents
- **SEC Security** — Ability to protect against deliberate destruction, damage and harm

## Availability

- **AVL Availability** — Ability to be operationally available anytime
- **UBQ Ubiquity** — Ability to be operationally present anywhere
- **RPT Repeatability** — Ability to repeat state and behaviour in sequence
- **RPR Reproducability** — Ability to reproduce state and behaviour from scratch
- **RCV Recoverability** — Ability to recover state and behaviour after a disastrous failure

• • •

## Quality

- **RLB Reliability** — Ability to perform required functions under stated conditions for a specified time
- **RSL Resilience** — Ability to provide an acceptable level of service in face of faults and challenges
- **RBS Robustness** — Ability to withstand stress, pressure, or changes in procedure or circumstances
- **STB Stability** — Ability to not suffer from internal failures in service
- **DRB Durability** — Ability to keep interfaces and functionality as is for a period of time
- **INT Integrity** — Ability to keep state consistency and avoid data corruption

## Performance

- **PRF Performance** — Ability to efficiently perform work, i.e., with a good work to time & resource ratio
- **SCL Scalability** — Ability to scale mostly linearly with changing requirements or conditions
- **RSP Responsiveness** — Ability to respond quickly to external interaction

## Structure

- **SMP Simplicity** — Ability to be plain, natural, straight-forward and with no observable complexity
- **FLX Flexibility** — Ability to be easily modifyable in order to respond to altered circumstances
- **MDL Modularity** — Ability to consist of individually comprehensible modules
- **ORT Orthogonality** — Ability to follow great separation of concerns in design

• • •

## Execution

- **EFF Efficiency** — Ability to perform work in the most economical way: good input/output ratio
- **EFC Efficacy** — Ability to perform work in order to getting things done and meeting targets
- **EFV Effectiveness** — Ability to perform the "right" work by setting right targets to achieve goals

## Interfacing

- **ITY Interoperability** — Ability to correctly operate and exchange information with foreign components
- **CPT Compatibility** — Ability to correctly operate despite expected older or newer interfaces

## Evolution

- **RSB Reusability** — Ability to reuse code or data with slight or no modifications
- **ADP Adaptability** — Ability to cope with smaller changes in the run-time environment
- **PRT Portability** — Ability to cope with larger changes in run-time environment
- **CFG Configurability** — Ability to individualize state and behaviour by non-destructive instructions
- **CST Customizability** — Ability to individualize state and behaviour by possibly destructive instructions
- **EXT Extensibility** — Ability to extend state and behaviour in a controlled way
- **TLR Tailorability** — Ability to adjust state and behaviour in a controlled way
- **MDF Modifiability** — Ability to change state and behaviour in an arbitrary way

• • •

Intellectual Content Version 1.0.0 (2020-08-13), Authored 2009–2020 by Dr. Ralf S. Engelschall, based on ISO/IEC FDIS 9126-1:2000
Graphical Illustration Version 1.1 (2019-07-18), © Copyright © 2011–2019 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved
Unauthorized Reproduction Prohibited, Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only

ARCHITECTURE FUNDAMENTALS

# Architecture Maxims

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 04.1

## Business Drives (BD)

Trigger and support the business with technological feasibilities, but always understand the business domain and its demands and align your architecture accordingly.



## Component Orientation (CO)

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.



## Separation of Business and Technology (BT)

Strictly separate the business, i.e., domain-specific, aspects from the technological, i.e., infrastructural, aspects. Furthermore, ensure the explicit visibility of domain concepts.



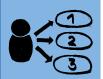
## Adequate Description (AD)

Provide as much stakeholder-directed architecture description as necessary, and as little as possible.



## Use-Case Driven Design (UC)

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.



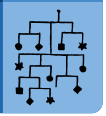

## Analytical and Creative Act (AC)

Recognize that every good architecture is based on both analytical engineering (structure) and creative artistic (beauty) aspects.



## Balance Principles Against Requirements (PR)

By weighing them against one another, find a reasonable balance between fundamental architecture principles and your particular non-functional requirements.



## Insights through Views & Aspects (VA)

Give insights into your architecture through carefully selected stakeholder-directed separate views and aspects. Express each with the most suitable graphical or textual language.



## Proven Basis (PB)

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 (TC)

Don't be too clever or tricky, both in your higher-level architecture and lower-level design aspects.



## Design for Failure Case (DF)

Murphy was an architect: everything which can fail will sometime ultimatively fail. Hence, already design for the failure case (think: "pessimistic").



## Continuous Compliance (CC)

Continuously check through qualitative inspections and quantitative measurements whether your architecture and the non-functional requirements are followed and do not drift apart.



## No Silver Bullet (SB)

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.



## Simplicity Trumps (ST)

Create solution parts as simple as possible and only as complex as necessary. And remember: simplicity before generality, use before reuse!



## Design to Change (DC)

Time changes everything, so your solution is already legacy at the first day of release. Hence, already design for its change (think: "agile").



## Integration-Figure Architect (IF)

Recognize that you, the architect, are the central integrating figure, having to bridge between the business and technology spheres of people.



## Stepwise Refinement (SR)

Start with the "big picture" and perform a stepwise top-down refinement of your architecture by going from coarse to fine aspects.



## Perfect is the Enemy of Good Enough (GE)

Beware of the perfection pitfall and design your architecture only as good as necessary and not as good as ultimately possible.




## Explicit Decisions (ED)

Record your major architecture decisions and rationales by taking into account and back-referencing the non-functional requirements.



## Eat Your Own Dog-Food (OF)

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.



Intellectual Content: Version 1.0.14 (2024-11-10), Authored 2009-2024 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.8 (2020-11-28), Copyright © 2012-2020 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved
Unauthorized Reproduction Prohibited. **Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.**

ARCHITECTURE FUNDAMENTALS

# Architecture Principles

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 04.2

## FL Factual Locality

Resources are as spatially and temporarily local-scoped to solution components as possible



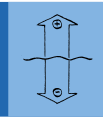
## ES Exclusive Sovereignty

Exclusive resource sovereignty by the enclosing component



## CA Contextual Adequacy

Neither insufficient nor exaggerated solutions for each context



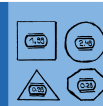
## SP Solution-oriented Proportionality

Good expected proportionality in each solution context



## HC Holistic Consistency

Full consistency across all aspects of a solution



## SH Structural Homogeneity

Maximum homogeneity in the structure of a solution



## CR Constructional Reusability

High reuse of proven structural components and partial solutions




## FS Fulfilled Standards

Compliancy to standards as much as possible, as long as the benefits predominate the drawbacks



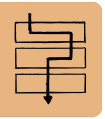
## FA Functional Abstraction

Suitable level of abstraction across all functional aspects of a solution



## FT Functional Traceability

Suitable traceability across all functional aspects of a solution



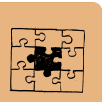
## CI Communicative Interoperability

Maximum interoperability in communication between solutions



## EH Environmental Harmony

Maximum harmony in the integration of the solution with its environment



## AR Avoided Redundancy

Minimum total number of copies of a single resource



## MS Minimum Special-Cases

Minimum total number of special-cases in a solution



## LS Logical Separation

Separation of concerns between the components of a solution



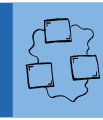
## SM Structural Modularity

Splitting of a solution into manageable structural components



## LC Loose Coupling

Loose coupling in communication and referencing between solution components



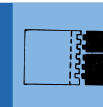
## SC Strong Cohesion

Strong relationship between functionalities within a single solution component



## OE Open Extensibility

Solution components can be extended by third-parties at fixed interfaces



## CC Closed Changeability

Solution components are protected against direct change by third-parties




## UI Unique Identification

Unique identification of all components of a solution



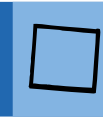

## UA Uniform Addressing

Uniform addressing of all components of a solution




## OS Overall Simplicity

All design aspects of a solution are as simple as possible and only as complicated as necessary



## EC Encapsulated Complexity

Complex related aspects of a solution are encapsulated into a single responsible component



## LA Least Astonishment

All design aspects of a solution are as little astonishing as possible and only as esoteric as necessary



## SD Self Documentation

All design aspects of a solution are preferably self-documenting



## OD Operational Delight

The solution provides users true delight even on long-term operation



## AA Artistic Aesthetics

The solution has holistic aesthetics and artistic love in details



Intellectual Content: Version 10.15 (2022-11-05), Authored 2010-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.5 (2018-08-10), Copyright © 2017-2018 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München / LMU for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Component Design

TECHNISCHE UNIVERSITÄT MÜNCHEN

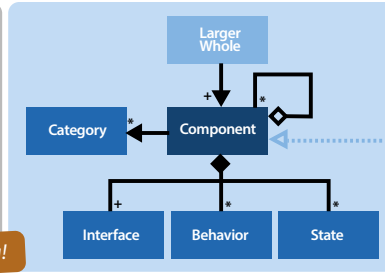
AF 05.1

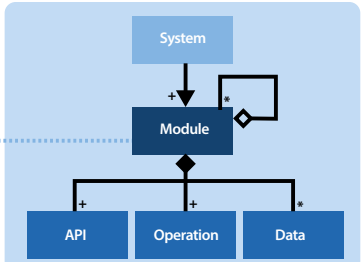
**Definition of a Component (of a Larger Whole):**
a **know-how encapsulating**, potentially **reusable** and **substitutable unit of hierarchical composition** with **explicit context dependencies**, which **hides the complexity** of its optional **behavior** and **state** realization behind **small contractually specified interfaces**, defines its **added value** in terms of **provided** and **consumed** interfaces and optionally belongs to zero or more **categories** of similar units.

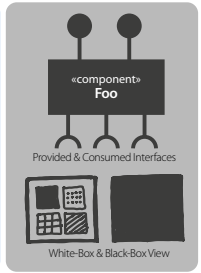
**Definition of a Module (of a System):**
a know-how encapsulating, potentially reusable and substitutable **source-code** unit of hierarchical composition with explicit context dependencies which hides the complexity of its **operation** and **data implementation** behind small contractually specified **Application Programming Interfaces (API)**, defines its added value in terms of provided and consumed **APIs** and optionally belongs to zero or more categories of similar units.

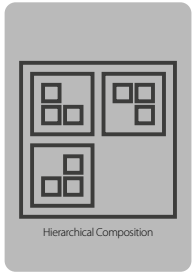
**Example Categories of Components:**
- Namespace
- Directory, File
- Configuration, Section, Directive
- Host, Virtual Machine, Container
- Process Group, Process, Thread
- Application, Microservice, Program
- Package, Class, Function
- Database, Schema, Table, Record
- Datamodel, Entity Group, Entity
- User Interface, Dialog, Widget

*Any group of anything!*




## How to find Components (or Modules)?

### DCA Domain Concept Abstraction
Model domain concepts as entity components and then group at higher levels.

### SOC Separation of Concerns
Build components for clearly distinct concerns.

### USE Reusability Potential
Decide on components based on their reusability potential.

### UCC Use-Case Clustering
Build domain components for each use-case or each logical use-case cluster.

### SRP Single Responsibility Principle
Build components for clearly distinct responsibilities.

### DCC Divide & Conquer Complexity
Master overall complexity by splitting larger things into smaller things.

### DDD Domain-Driven Design
Model domain "Bounded Contexts" through DDD and derive components from them.

### CNC Coupling and Cohesion
Decide on components based on their loose coupling and strong cohesion.

### CCC Cross-Cutting Concerns
Build common cross-cutting concerns as cross-cutting components.

### OOD Object-Oriented Design
Model Object-Oriented Design entities (and/or OOP constructs) as components.

### DEP Dependency Encapsulation
Decide on components based on their encapsulation of dependencies.

### PAT Architecture Patterns
Build inner components to comply to outer structure, slicing and clustering architectures.

Intellectual Content: Version 1.0.1 (2020-09-08), Authored 2017-2020 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.7 (2021-10-23), Copyright © 2017-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. **Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.**

ARCHITECTURE FUNDAMENTALS

# Interface Design

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 05.2

**Definition of an Interface:** well-defined shielding and abstracting **boundary** of a passive, providing **component**, consisting of one or more distinguished, **outside-in** designed, **interaction endpoints**, each accessed and controlled by active, consuming components through the **exchange** of **input/output information** and operating under a certain **syntactical** and **semantical contract.**



**Endpoint:** Name, Directive, Command, Function, Method, Procedure, Address, Port, URL, Dialog, ...

**Exchange:** Option, Argument, Parameter, Return Value, Result, Request/Response Message, Error/Exception, Interaction, ...

**Contract:** Syntax, Pre-Condition, Invariant, Post-Condition, Side-Effect, Idempotence, Determinism, Functionality, ...

## Types of Software Interfaces

### API Application Programming Interface

Example: `foo("bar", 42)` (call and use)




### SPI Service Provider Interface

Example: `register("foo", (x,k) => ...)` (extend and implement)




### SCI Startup Configuration Interface

Examples: INI, Java Properties, TOML, YAML, JSON, XML, etc.



### BPI Batch Processing Interface

Examples: Unix at(1), Unix ts(1), GNU Batch, Spring Batch, Java Batch, SAP LO-BM, etc.



### CLI Command-Line Interface

Example: `foo -x --bar=baz quux`



### GUI Graphical User Interface

Examples: Windows/WPF, macOS/Cocoa, KDE/Qt, GNOME/GTK




### RNI Remote Network Interface

Examples: GraphQL-IO, HTTP/REST, SOAP, RMI, WebSockets, AMQP, MQTT, etc.



## Characteristics of Good Interfaces

### AP Appropriate & Proportional

Appropriate to consumer requirements, proportional to provider functionality.



### SA Shielding & Abstracting

Shields from direct access, abstracts and hides implementation details.



### IE Inviting & Expressive

Invites through "outside – in" design, powerful in expressiveness.



### IF Intuitive & Foolproof

Intuitive to grasp and use, hard to misuse.



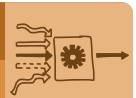
### OC Orthogonal & Concise

Supports combinatorial use-cases, causes minimum boilerplate.



### TP Tolerant & Predictable

Tolerant on input, predictable on output.



### EC Extensible & Compatible

Easy to extend for providers, backward/forward-compatible for consumers.




## Selected Interface Design Patterns

### IVF Interface Version & Features

Provide version and feature information for algebraic comparison and feature detection.




### 2LF Leaky Two-Layer Facade

Provide higher-level convenient use-case and lower-level orthogonal feature interface.



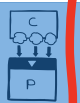
### EVE Event Emitter

Emit events to previously registered, interested consumers.



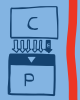
### CTX Multi-Context

Use contexts to distinguish between different usage scenarios and to carry common info.




### CEF Configure-Execute Flow

Spread use-cases onto a flow of configuration exchanges and a final executional exchange.




### IOC Inversion Of Control

Invert control of asynchronous operations via callbacks, promises or async. mechanisms.




### HMR Human/Machine Responses

Support humans and machines in outputs through both description and parsing-free info.

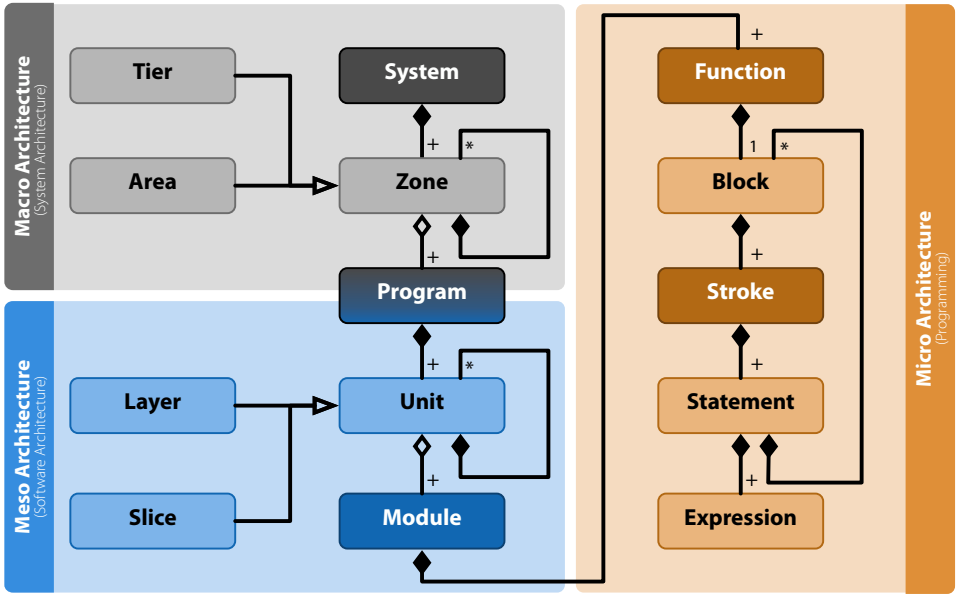


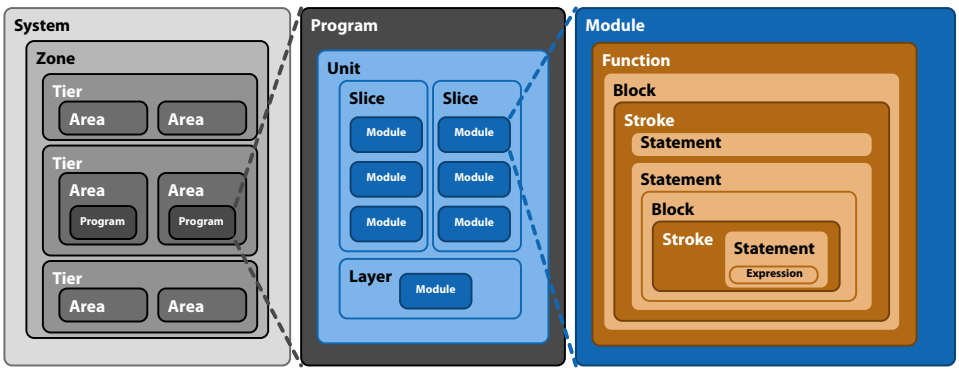

Intellectual Content: Version 1.0.9 (2022-11-05) Authored 2015-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.8 (2022-11-05) Copyright © 2015-2022 Dr. Ralf S. Engelschall <http://engelschall.com> All Rights Reserved
Unauthorized Reproduction Prohibited. **Licensed to Technische Universität München/Fakultät für Informatik for reproduction in Computer Science Lecture contexts only.**

ARCHITECTURE FUNDAMENTALS

# Component Hierarchy

TECHNISCHE UNIVERSITÄT MÜNCHEN




**Zone:**
Structuring **Tier** or **Area** within a **System** (or within an outer **Zone**).

**Unit:**
Structuring **Layer** or **Slice** within a **Program** (or within an outer **Unit**).

**Tier:**
Special kind of ranked **Zone** for horizontally grouping (logically or spatially distinct and network-linked) **Programs** or other **Tiers** and **Areas**.

**Layer:**
Special kind of ranked **Unit** for horizontally grouping (logically distinct and memory-linked) **Modules** or other **Slices** and **Layers**.

**Area:**
Special kind of unranked **Zone** for vertically grouping (logically or spatially distinct and network-linked) **Programs** or other **Tiers** and **Areas**.

**Slice:**
Special kind of unranked **Unit** for vertically grouping (logically distinct and memory-linked) **Modules** or other **Slices** and **Layers**.

**System:**
One or more network-linked **Programs**, forming a solution as a whole and the primary unit of thinking at the level of Macro Architecture.

**Module:**
Know-how encapsulating, potentially reusable group of **Functions** and primary unit of thinking at the level of Meso Architecture.

**Program:**
Stand-alone, self-contained, executable piece of software and primary unit of thinking at the level of both Macro and Meso Architecture.

**Function:**
Operation of a **Module**, with a well-defined interface, and one of the two primary units of thinking at the level of Micro Architecture.

**Statement:**
Execution of a single action, specified through one or more **Expressions** and zero or more inner **Blocks**.

**Stroke:**
Logical sequence of **Statements** in a **Block** and one of the two primary units of thinking at the level of Micro Architecture.

**Block:**
Sequence of one or more **Strokes** within a **Function** or an outer **Statement** and scope for variables of inner **Statements**.

**Expression:**
Simple or complex representation of a single data value.

AF 05.3

Intellectual Content: Version 1.2.3 (2022-11-05) Authored 2018-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.2.1 (2020-07-08) Copyright © 2018-2020 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Architecture Patterns

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 06.1

## Pattern Definition

**Pattern:** unique, catchy, and associative **Name** and concise, precise, and normative **Description** of a contextual, regularly occurring, and practically relevant **Problem** and a general, highly reusable, and best practice **Solution** for it.



## Pattern Structure

Pattern

| | | |
|---|---|---|
| Name | unique | has to be unambiguously distinguished |
| | catchy | has to be recognized and memorized |
| | associative | has to be intuitively associated to the solution |
| Description | concise | has to be reasonably remembered |
| | precise | has to be clear and unambiguous |
| | normative | has to be foundational and stringent in expressiveness |
| **Problem** | contextual | has to be described in a given context |
| | regularly occurring | has to be motivated by enough situations |
| | practically relevant | has to be motivated by enough practice |
| **Solution** | general | has to be general enough to be sufficiently reusable |
| | highly reusable | has to be applicable also in variants of the context |
| | best practice | has to be considered a best or at least good practice |

## Pattern Rationale

Intellectual Content: Version 1.0.0 (2022-11-05) Authored 2022 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.0.0 (2022-11-05) Copyright © 2022 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Layer Architectures

(Patterns)

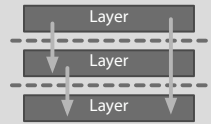
TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 06.2

## Layering Principle

Horizontally split code or data into two or more logically, optionally also spatially, clearly distinct, isolating, named, and ranked Layers.

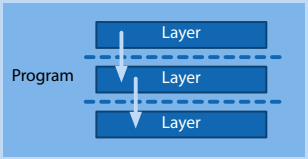
A Layer is not allowed to have relationships to or knowledge about any upper Layers. Additionally, for *Closed Layering*, each Layer is allowed to have relationships to and knowledge about the *directly* lower Layer only. In contrast to *Open Layering* or *Leaky Abstraction*, where each Layer is allowed to have relationships to and knowledge about *any* lower Layer.




## LR Layer

Split related code or data of a Program into two or more logically distinct domain- or technology-induced Layers.

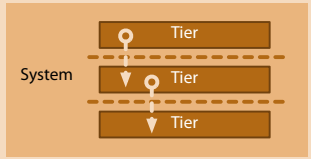
Rationale: Separation of Concern, Single Responsibility Principle, Mastering Complexity, Change Isolation, Functional Abstraction.




## TR Tier

Split related code or data of a System into two, three or more logically and spatially distinct, network-connected, domain- or technology-induced Tiers.

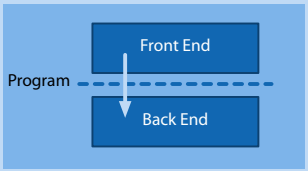
Rationale: Separation of Concern, Single Responsibility Principle, Mastering Complexity, Change Isolation, Functional Abstraction, Deployment Partitioning.




## FB Front End / Back End

Split the code of a Program into exactly two logical Layers: a user-facing Front End and a data-facing Back End.

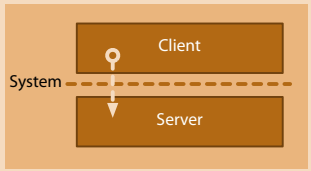
Rationale: Separation of Concern, Single Responsibility Principle, Mastering Complexity, Change Isolation, Functional Abstraction, Organisational Alignment.




## CS Client / Server

Split the code of a System into two spatially distinct, network-connected Layers, each forming a stand-alone Program: a user-facing and multi-instantiated (Rich) Client and a data-facing (and logically) single-instantiated (Thin) Server. Both contain a Front/Back End.

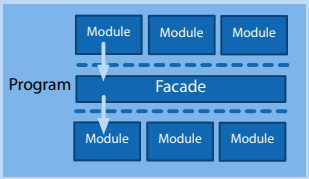
Rationale: Multi-User, User Computing Resource Leverage, Distributed Computing.




## FD Facade

Splice a domain-specific Facade Layer into two Layers of two or more Modules. The extra Facade Layer acts as a broker between the Modules.

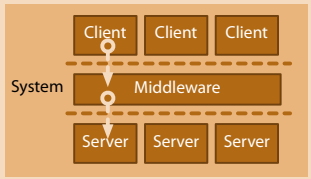
Rationale: Information Hiding, Cross-Cutting Concern Centralization, Functionality Orchestration, Authorization, Validation, Conversion.




## MW Middleware

Splice a domain-unspecific Middleware Layer into a Client/Server communication. The extra Layer is a stand-alone Program Tier and acts as a broker between Client and Server.

Rationale: Communication Peer Discovery Simplification, Transport Protocol Conversions, Network Topology Flexibility.




Intellectual Content: Version 1.0.5 (2020-11-28), Authored 2018-2020 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.4 (2020-11-28), Copyright © 2018-2020 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Slice Architectures

(Patterns)

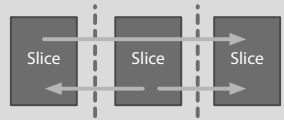
TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 06.3

## Slicing Principle

Vertically split code or data into two or more logically, optionally also spatially, clearly distinct, named, and unranked slices.

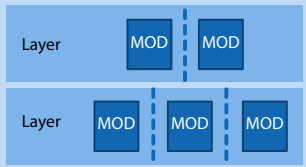
The particular slicing should minimize the total amount of individual relationships between the resulting slices. Per type of relationship, there should be no cycle in the transitive relationships.




## MOD Concerned Module

Split related code or data (usually across a single Layer) into two or more logically distinct domain- or technology-induced Modules.

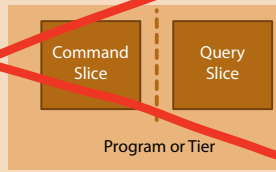
Rationale: Separation of Concern, Single Responsibility Principle, Mastering Complexity.




## CQRS Command-Query Responsibility Segregation

Split code and data of a Program (across all Layers) or a Tier into exactly two slices to segregate operations that read data (queries) from the operations that update data (commands).

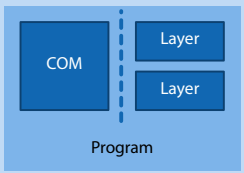
Rationale: Separated Scalability, Separated Data Access Patterns, Event Sourcing Approach.




## COM Common Slice

Factor out common or cross concern code or data of a Program (across all Layers) into a single spatially distinct, separate slice.

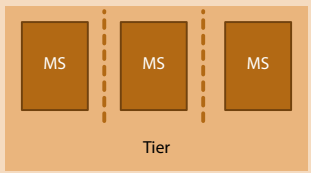
Rationale: Lack of Redundancy, Single Point of Truth, Reusability.




## MS Microservice

Split code and data of a Tier (across all Layers) into two or more distinct, loosely-coupled, domain-enclosed, functional services, each forming a stand-alone Program.

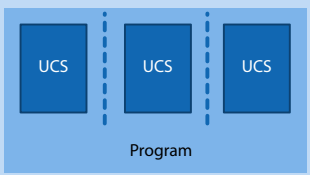
Rationale: Heterogeneity, Long-Term Large-Scale Maintenance, Replaceability, Resilience, (Scalability), (Easy Deployment), (Organizational Alignment), (Composability), (Reusability).




## UCS Use-Case Slice

Split the code and data of a Program (across all Layers) into two or more purely logical slices, one for each distinct, domain-specific Use-Case.

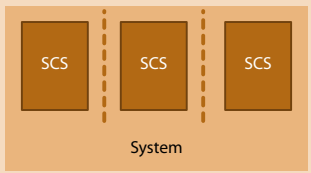
Rationale: Comprehensibility, Domain Alignment, Mastering Complexity.




## SCS Self-Contained System

Split code and data of a System (across all Layers and Tiers) into two or more distinct, loosely-coupled, domain-enclosed, functional systems, each forming a stand-alone sub-System.

Rationale: Mastering Complexity, Heterogeneity, Resilience, Scalability, Easy Deployment, Organizational Alignment, Reusability, Replaceability.




Intellectual Content: Version 1.0.5 (2024-04-15) Authored 2018-2024 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.2 (2019-09-10) Copyright © 2018-2019 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Flow Architectures

(Patterns)

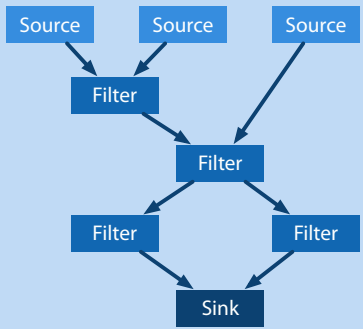
TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 07.1

## Pipes & Filters

Pass data through a directed graph of **Components** and connecting **Pipes**. The components can be **Sources**, where data is produced, **Filters**, where data is processed, or **Sinks**, where data is captured. Source and Filter components can have one or more output Pipes. Filter and Sink components can have one or more input Pipes. Components are independent processing units and operate fully asynchronously.

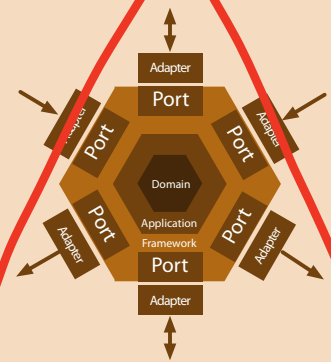
Examples: Unix commands with stdin/stdout/stderr and the Unix shell connecting them with pipes; Apache Spark or Apache Camel data stream processing pipelines.




## Ports & Adapters (Hexagonal)

Perform communication in a Hub & Spoke fashion by structuring a solution into the three "Layers" **Domain**, **Application** and **Framework** and use the Framework layer to connect with the outside through **Ports** (general Interfaces) and **Adapters** (particular implementations). Often some Ports & Adapters are user-facing sources and some are data-facing sinks, although the motivation for the Ports & Adapters architecture is to remove this distinction between user and data sides of a solution.

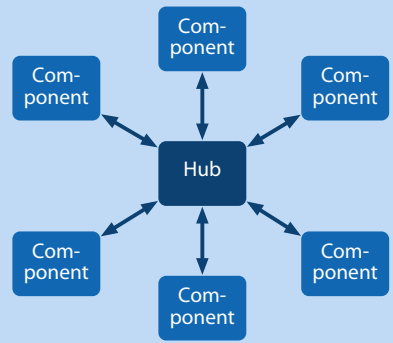
Examples: Message Queue, Enterprise Service Bus or Media Streaming Service internal realization.




## Hub & Spoke

Perform communication (the **Spoke**) between multiple Components through a central **Hub** Component. Instead of having to communicate with N x (N-1) / 2 bi-directional interconnects between N Components, use the intermediate Hub to communicate with just N interconnects only. Sometimes one distinguishes between K (0 < K < N) source and N - K target Components and then K x (N - K) uni-directional interconnects are reduced to just N interconnects, too.

Examples: Message Queue, Enterprise Service Bus, Module Group Facade, GNU Compiler Collection, ImageMagick, etc.




Intellectual Content: Version 1.0.2 (2018-10-29) Authored 2018 by Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Graphical Illustration: Version 1.0.3 (2018-11-24) Copyright © 2018 Dr. Ralf S. Engelschall. Unauthorized reproduction prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Process Architectures

(Patterns)

TECHNISCHE UNIVERSITÄT MÜNCHEN

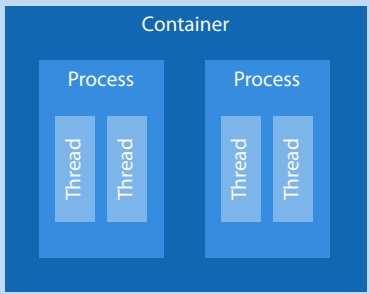
AF 07.2

## Container, Process, Thread

The Operating System manages and orchestrates the run-time execution of applications in **Containers**, programs in **Processes** and control flows in **Threads**.

Containers are the ultimate enclosures, separating and controlling both the computing resources processor, memory, storage and network. Processes are the primary enclosures, still separating and controlling at least the computing resources processor and memory. Threads are the light-weight enclosures, just separating and controlling the computing resource processor. Containers can contain one or more Processes, and Processes can contain one or more Threads.

Examples: Docker Container, Unix Processes, POSIX Threads.

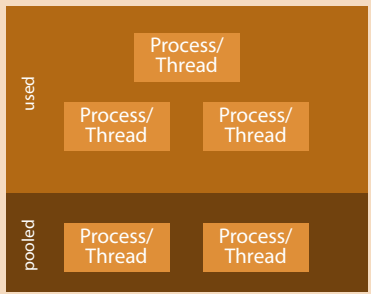



## Process/Thread Pool

Instead of creating a Process/Thread for handling each incoming I/O request, pick a pre-created Process/Thread out of a resource **Pool** in order to increase performance and decouple I/O traffic (leading to threads of execution) from the actual computing resource usage and utilization.

The Process/Thread Pool usually has a lower and upper bound of processes/threads. The lower bound keeps the system "hot" between I/O requests. The upper bound limits the computing resource usage and avoids over-utilization.

Examples: Apache HTTP Daemon

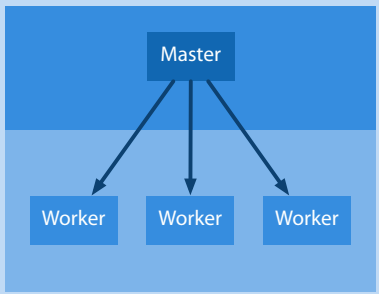



## Master-Worker

The system has a single permanent **Master** container/process/thread and a Pool of many ephemeral **Worker** containers/processes/threads. The Master starts, restarts, pauses, resumes and stops the Workers and usually also delegates incoming I/O requests to them. The Workers process the I/O requests and deliver the responses.

Starting the Master usually implicitly starts an initial set of Workers (the initial Pool), stopping the Master implicitly stops all still pending Workers.

Examples: Unix init(8) daemon, Apache HTTP Daemon, SupervisorD, Node.js Cluster module




Intellectual Content: Version 1.0.2 (2019-09-21) Authored 2018-2019 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.1 (2018-11-24) Copyright © 2018 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Cluster Architectures

(Patterns)

TECHNISCHE UNIVERSITÄT MÜNCHEN

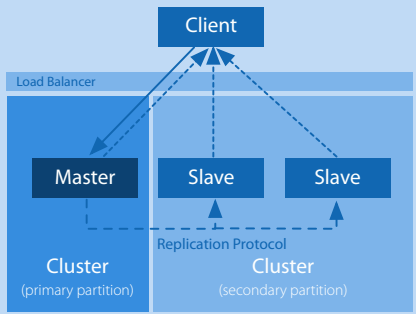
AF 07.3

## Master-Slave (Static Replication)

Cluster of a single **Master** and multiple **Slave** nodes, where data is continuously copied from the Master to the Slave nodes in order to support high-availability (where a Slave will take over the Master role) in case of a Master outage and increased read performance (where regular read requests are also served by the Slaves).

In this static replication scenario the Master is usually assigned statically and in case of outages has to be reassigned usually semi-manually. Especially, the full reestablishment of the original Master assignment after a Master recovery usually is a manual process.

Examples: OpenLDAP Replication, PostgreSQL WAL Replication.

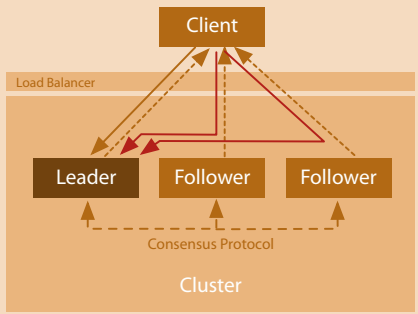



## Leader-Follower (Dynamic Replication)

Cluster of a single **Leader** and multiple **Follower** nodes, where data is written on the current Leader node and data read on both the current Leader and all Follower nodes. For writing data to the cluster, the Leader node performs a consensus protocol (e.g. RAFT, Paxos or at least Two-Phase-Commit) with the Followers and this way automatically and consistently replicates the data to the Followers.

In this dynamic replication scenario the Leader is usually automatically assigned by the cluster nodes through an election protocol and in case of outages is automatically re-assigned. There is usually no re-establishment of the original Leader assignment.

Examples: Apache Zookeeper, Consul, EtcD, CockroachDB, InfluxDB.

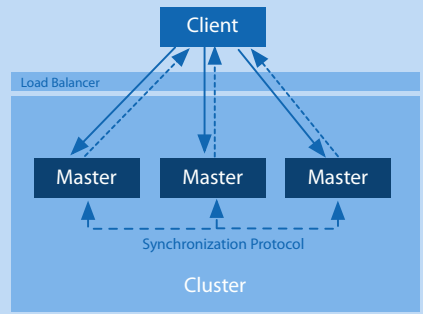


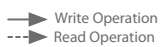

## Master-Master (Synchronization)

Cluster of multiple **Master** nodes, where data is read and written on any Master node concurrently. The Master nodes either use Strict Consistency through writing to a mutual-exclusion-locked shared storage concurrently or use Eventual Consistency in a Shared Nothing storage scenario where they continuously synchronize their local data state to all other nodes with the help of a synchronization protocol.

The synchronization protocol usually is based on either Conflict-Free Replicated Data Types (CRDT) or at least Operational Transformation (OT). In any scenario, data update conflicts are explicitly avoided.

Examples: ORACLE RAC, MySQL/MariaDB Galera Cluster, Riak, Automerge/Hypermerge.




Intellectual Content: Version 1.0.5 (2021-09-08), Authored 2018-2021 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.0.4 (2021-09-28), Copyright © 2018-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited, Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Networking Architectures

(Patterns)

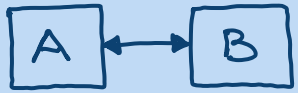
TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 08.1

## PTP Point-to-Point

Communicate between two network nodes in a point-to-point fashion, usually through a direct link.

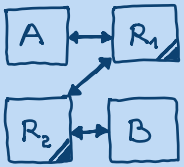
**Rationale:** simple communication where both nodes know about each other and can directly reach each other.




## RTG Routing

Communicate between two network nodes in a point-to-point fashion, but by routing the network packets over intermediate forwarding nodes (routers).

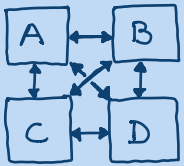
**Rationale:** simple communication where both nodes know about each other, but cannot directly reach each other.




## P2P Peer-to-Peer

Communicate between multiple network nodes (usually all in the client and server role at the same time) without involving a central hub node (in the role of a server) — except for the initial network entry discovery.

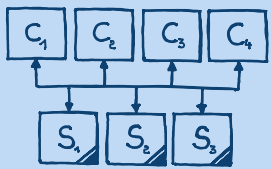
**Rationale:** communication without central control (although a seed peer is required).




## C/S Client/Server

Communicate between multiple nodes in the client role (making requests, and usually with ephemeral addresses) and multiple nodes in the server role (serving responses, and usually with fixed addresses).

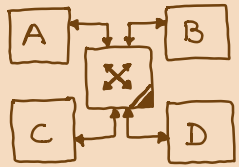
**Rationale:** communication with central orchestration, control and data storage.




## BUS Bus/Broker/Relay

Communicate between multiple nodes with the help of a central packet forwarding hub node in a star network topology.

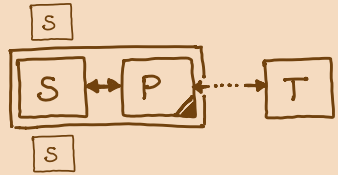
**Rationale:** decouple communication nodes: instead of Point-to-Point (PTP) communications between all nodes, there are just PTP communications with the hub.




## FPR (Forward) Proxy

Communicate between two nodes by using an intermediate forwarding proxy node in front of the source node.

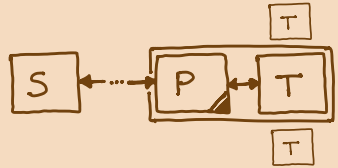
**Rationale:** bridge network topology constraints (segmented networks); caching at source side; auditing of communication.




## RPR Reverse Proxy

Communicate between a source and a target node by using a masquerading proxy node directly in front of the target node.

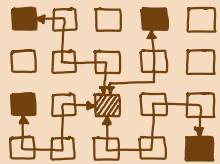
**Rationale:** load balancing for multiple target nodes; caching at target side; auditing of communication; security shielding of target nodes; protocol conversions.




## VPN Virtual (Private) Network

Communicate between nodes in a logical star network topology on top of an arbitrary physical routed network topology.

**Rationale:** secure private network overlaying an unsecure public network; simplify network topology.



Intellectual Content: Version 1.0.7 (2022-10-09), Authored 2018-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.4 (2019-10-31) Copyright © 2018-2019 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Communication Architectures

(Patterns)

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 08.2

## UCT Unicast (one-to-one)

Communicate messages from one source to exactly one destination node. The destination node is explicitly and individually addressed.

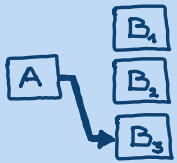
**Rationale:** private communication between exactly two nodes which both know each other beforehand.




## ACT Anycast (one-to-any)

Communicate messages from one source to one of many destination nodes. The picked destination node usually is the network-topology-wise "nearest" or least utilized node in a group of nodes.

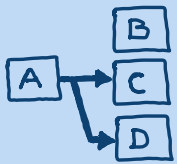
**Rationale:** Unicast, optimized for network failover scenarios, load balancing and CDNs.




## MCT Multicast (one-to-many)

Communicate messages from one source to many destination nodes. The destination nodes usually form a group and are usually not individually addressed.

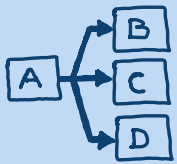
**Rationale:** node communication where destination nodes dynamically change or where total traffic should be reduced.




## BCT Broadcast (one-to-all)

Communicate messages from one source to all available destination nodes. The destination nodes usually are implicitly defined by the extend of the local communication network segment.

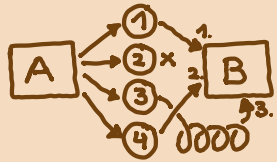
**Rationale:** spreading out messages to all available nodes for potential responses.




## DGR Datagram (Single Packet)

Communicate messages as an unordered set of single packets, usually without any network congestion control, retries or other delivery guarantees.

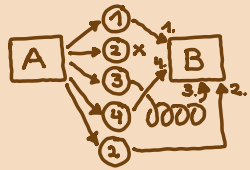
**Rationale:** simple low-overhead communication without prior communication establishment (handshake).




## STR Stream (Sequence of Packets)

Communicate messages as an ordered sequence (stream) of packets, usually with network congestion control, retries and delivery guarantees (at-most-once, exactly-once, at-least-once).

**Rationale:** reliable communication between nodes.




## PLL Pull (Request/Response, RPC)

Communicate by performing a request (from the client node) and pulling a corresponding response (from the server node).

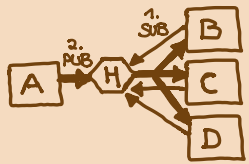
**Rationale:** Remote Procedure Call (RPC) like Unicast or Anycast communication.




## PSH Push (Publish/Subscribe, Events)

Communicate by "subscribing" to "channels" of messages (on one or more receiver nodes or on an intermediate hub) once and then publishing events to those "channels" (on the sender node) multiple times.

**Rationale:** event-based Multicast or Broadcast communication.




Intellectual Content: Version 1.0.3 (2020-09-26), Authored 2018-2020 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.0.1 (2018-11-24), Copyright © 2018 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Data Structure Architectures

(Patterns)

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 09.1

## Data Structure Types

**Scalar,** Atom, Primitive Type

Plain integer or real number, single character or character string, not indexed and (for string only) accessed in O(1) by character position.

**Tuple,** Object, Structural Type, Record

Ordered, fixed-size sequence of scalar elements, each of individual type, indexed by name and accessed in O(1) by element name.

**Sequence,** Array, List

Ordered sequence of elements, each of same type, indexed by position and accessed in O(1) or O(n) by element position.

**Set,** Bag, Bucket

Unordered set of elements, each of same type, not indexed and accessed in O(1) or O(n) by element reference.

**Map,** Hash, Associative Array

Unordered sequence of elements, each of same type, indexed by (scalar) key and accessed in O(1) by key.

**Graph,** Nodes & Edges

Unordered set of linked elements (nodes), each of individual type, indexed by (scalar) key and accessed in O(1) by key or by following a directed link (edge).

## Data Evolution Approaches

### In-Place Editing

Modify data through direct in-place editing, overwriting the previous revision.

### Stacking Revisions

Modify data through stacking revisions, preserving all previous revisions. Latest revision is always on top of stack.

### Structural Difference

Modify data through merging, journaled domain-unspecific structural differences.

### Operational Transformation (OT)

Modify data through applying journaled, domain-specific operational transformations.

## Data Sharing Approaches

### Event Sourcing & CRDT

Share data as chronological sequence of data change events from which the data states can be (re)constructed. Optionally, use a Conflict-Free Replicated Data-Type (CRDT) protocol for the change events.

### Ref.-Counting & Copy-on-Write

Share data between resources by using reference-counted data chunks, duplicating a chunk (and resetting its reference count to one) on write operations only and destroying a chunk once the reference count drops to zero.

## Data Store Types

### Key-Value Store

Storage of values in an unordered manner, indexed and queried by key.

Redis, Riak, MemcacheD, RocksDB, LevelDB

### Large-Object Store

Storage of unstructured binary-large object (BLOB) data and its associated meta-data, indexed and queried by unique id.

Minio, SeaweedFS, AWS S3

### Triple Store

Storage of subject-predicate-object triples, indexed and queried by subject/predicate/object values and example triples.

Redstore, Virtuoso

### File-Tree Store

Storage of unstructured data as named files in a directory tree, indexed and queried by name path from root directory to leave file.

ZFS, XFS, UFS2, ADFS

### Graph Store

Storage of values as vertices and edges in a graph, both optionally referencing associated key/value pairs. Indexed and queried by key/value pairs and traversed by following edges.

Neo4J, OrientDB, ArangoDB

### Document Store

Storage of structured "documents", indexed by id and key/value fields and queried by id and example documents.

MongoDB, CouchDB, RethinkDB

### Relational/Table Store

Storage rows of fixed-size, typed value columns, indexed and queried by column values.

PostgreSQL, MariaDB, SQLite, H2, ORACLE DB, IBM DB2

### Full-Text Store

Storage of unstructured text, indexed and queried by content words.

ElasticSearch, Solr, Groonga

### Wide-Column Store

Distributed storage of rows of sparse (often untyped) value columns, indexed and queried by column values.

Cassandra, MonetDB, HBase, ScyllaDB

### Time-Series Store

InfluxDB, MetricTank, RRDTool

Storage of integer or real values (y-axis) of a time-series (x-axis) into a fixed-size storage format in a round-robin manner where older values are increasingly aggregated (leading to lower resolutions at older times) and finally overwritten.

### DataVault Store

Long-term historical storage of foreign, arbitrary relational data in a fixed schema of hubs, links and satellites, indexed and queried for analysis and reporting purposes.

DataVault 2.0

### BlockChain Store

Storage of values in an unordered manner within information blocks which are cryptographically chained through their hash values and distributed in a peer-to-peer way.

Ethereum, Quorum, Tendermint, Hyperledger

Intellectual Content: Version 1.1.3 (2020-09-26), Authored 2011-2020 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.1.2 (2013-07-28), Copyright © 2012-2020 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Data Persistence Architectures

(Patterns)

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 09.2

## Data Guarantees

### CAP (Trade-In)

A distributed data store cannot provide more than two out of three guarantees: Consistency (C), Availability (A), Partition-Tolerance (P). So, it has to choose between Consistency (CP) and Availability (AP) when a network partition or failure happens.




### BASE (NoSQL)

The semantics (usually of NoSQL systems) of (B)asically (A)vailable, (S)oft state, and (E)ventual consistency. BASE systems favor Availability over Consistency in the CAP-context.




### ACID (RDBMS, NewSQL)

The four guarantees provided in parallel (usually by RDBMS and NewSQL systems): Atomicity, Consistency, Isolation and Durability. ACID systems usually favor Consistency over Availability in the CAP-context.




## Data Access

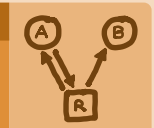
### Shared Read/Write

Shared access to data for both read and write operations. Example: Multiple threads on heap or Master-Master database setup.



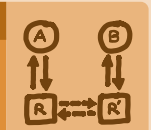

### Shared Read / Exclusive Write

Shared access to data for read operations and exclusive access (via a single "owning" component) to data for write operations. Example: RDBMS Master-Slave cluster with shared storage.




### Shared Nothing

No shared access to data at all for both read and write operations. Example: Leader-Follower setup with RAFT consensus where Leader writes data only.




## Data Spreading & Aggregation

### Data River (1-to-N)

A real-time fan-out replication of data from a single upstream/source data repository to multiple downstream/target data repositories.



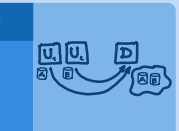

### Data Mart (N-to-1), ODS

A massive sized, easily accessible data repository for storing "big data" from many upstream sources in a (real-time and) structured way and with knowing the actual subsequent analysis usage.




### Data Lake (N-to-1), Cache

A massive sized, easily accessible data repository for storing "big data" from many upstream sources in a (real-time) semi-structured way and without knowing the actual subsequent usage.




## Data Access Grouping

### Transaction

Protect a sequence of operations from interim exceptions by bracketing the operations in a technical transaction (ensuring that either all or none of the operations succeed).



### Compensation

Protect a sequence of operations from interim exceptions by undoing the already succeeded operations through domain-specific compensating (reverse) operations.



## Data Consistency

### Exclusive Locking (Mutex)

Protect data from concurrent access and resulting inconsistencies with a mutual exclusion lock (mutex) which allows just a single peer to access the data at a time.




### Optimistic Locking

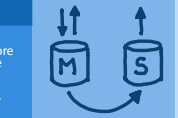
Protect data from concurrent access and resulting inconsistencies by taking note of a revision number or content hash during read operations and checking that this information has not changed before writing the data.




## Data Transfer

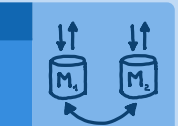
### Replication

Continuously stream or regularly copy data from a master system to one or more slave systems in order to either read the data from slave systems faster or have slave systems available as a fallback/backup in case of a failure of the master system.




### Synchronization

Continuously stream or regularly copy data between multiple master systems and resolve potential concurrent data modification conflicts. This way allow distributed and even disconnected computing.




Intellectual Content: Version 1.0.2 (2019-04-12), Authored (2018-2019) by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.1.0 (2021-11-06), Copyright © 2018-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

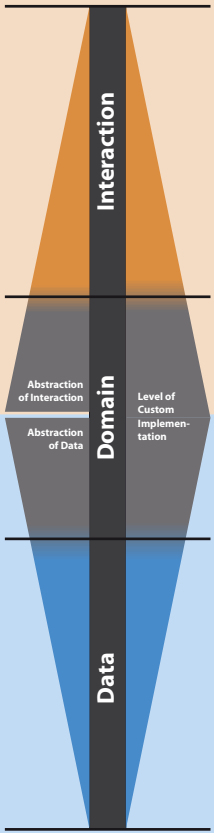
# Application Reference Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

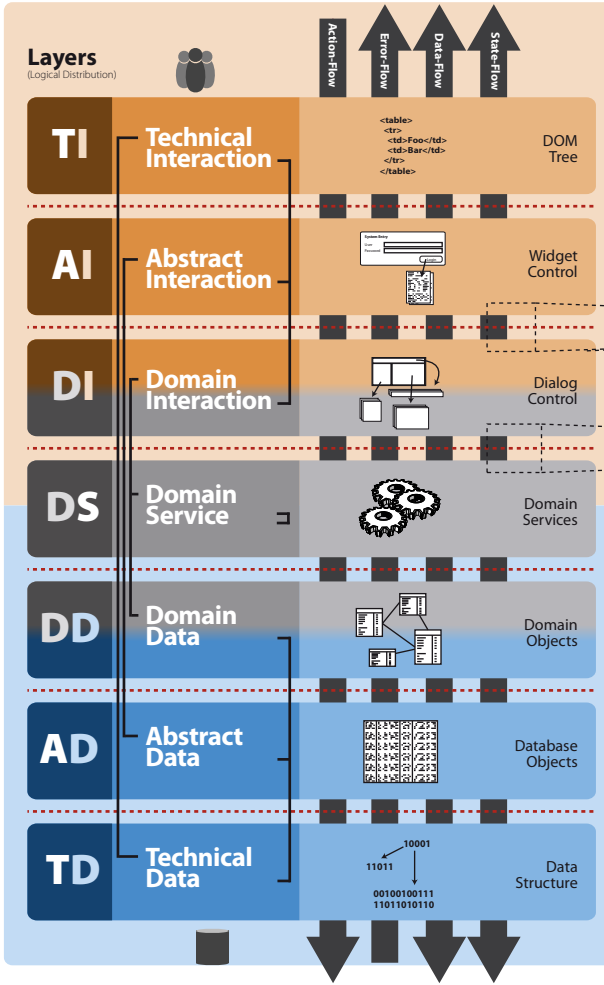
AF 10.1

Intellectual Content: Version 10.0 (2022-10-09), Authored 2009-2022 by Matthias Bruderlein, Andreas Martens and Dr. Ralf S. Engelschall.
Graphical Illustration: Version 1.0 (2022-10-09), Copyright © 2012-2022 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contents only.

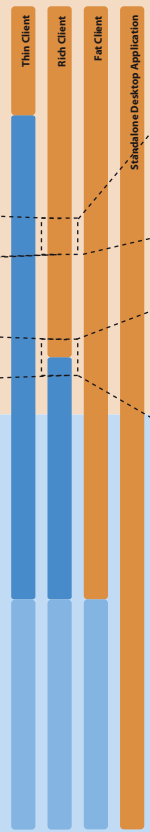
## Focus
(Abstraction & Implementation)

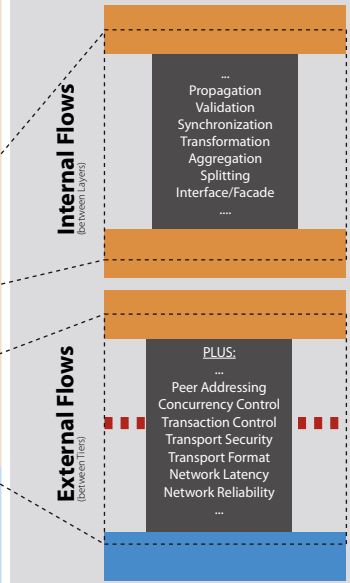



## Layers
(Logical Distribution)

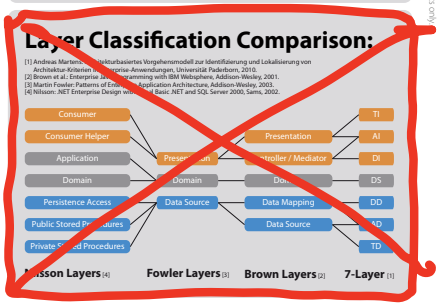



## Tiers
(Physical Distribution)




## Layer Classification Comparison:



ARCHITECTURE FUNDAMENTALS

# Client-Server Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 10.2

Intellectual Content: Version 1.00 (2021-09-14), Authored 2021 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.00 (2021-09-14), Copyright © 2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

## Thin-Client Architecture

**Thin-Client Architecture:** the application consists of a single custom **Server** program **only**. The **Server** program renders the UI mask and keeps the entire UI state on the **Server**.



Client Run-Time
(Chrome, Firefox, etc)

User Interface Mask
(HTML/SVG/PNG/CSS)

Network Protocol
(HTTP only)

User Interface Mask
(HTML/SVG/PNG/CSS)

Server Program

Interaction Layer — UI

Service Layer — AC

Data Layer — DP

Front End

Back End

Server Run-Time
(JVM, Node.js, etc)



Examples:
PHP, JSP, JSF, Next.js,
ASP.NET, BSP, etc.

**UI:** User Interface
**AC:** Application Core
**DP:** Data Persistance

## Rich-Client Architecture

**Rich-Client Architecture:** the application consists of both custom **Client and Server** programs. The **Client** program renders the UI mask and keeps the entire UI state on the **Client**.



Client Run-Time
(Chrome, Firefox, etc)

User Interface Mask
(HTML/SVG/PNG/CSS)

Client Program

Interaction Layer — UI

Service Layer — AC

Data Layer — RPC/C

Front End

Back End

Domain Data
(JSON/XML)

Network Protocol
(HTTP, WebSocket,
or WebRTC)

Server Program

Interaction Layer — RPC/S

Service Layer — AC

Data Layer — DP

Front End

Back End

Server Run-Time
(JVM, Node.js, etc)



**RPC/C:** Remote Procedure Call / Client-Side
**RPC/S:** Remote Procedure Call / Server-Side

Examples:
W3C HTML5 (Angular, React, Vue),
Flash/Flex, Eclipse RCP, etc.

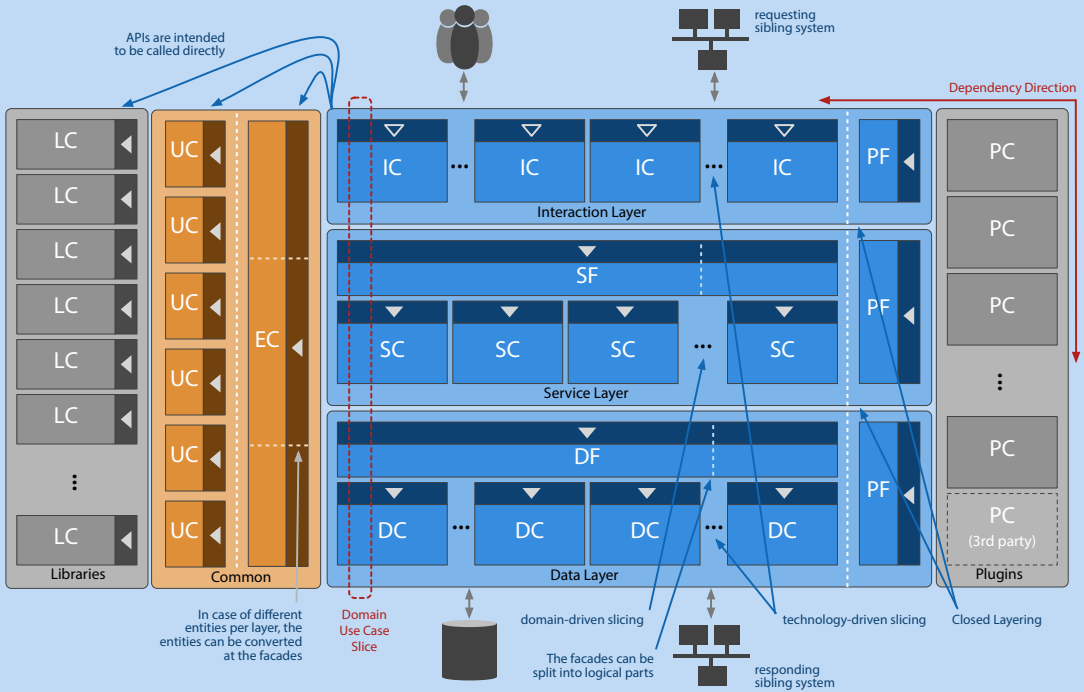
ARCHITECTURE FUNDAMENTALS

# Information System Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 10.3

## Reference Architecture Blueprint




**Component Types:**

IC: Interaction Component
SF: Service Facade
SC: Service Component
DF: Data Facade
DC: Data Component
PF: Plugin Facade
PC: Plugin Component
UC: Utility Component
EC: Entity Component
LC: Library Component

**Service Facade (SF) Potential Functionality:**
Request Validation, Use-Case Authorization, Process Orchestration (Dispatching), Data Splitting/Aggregation/Conversion, Transaction Handling, Run-Time Use-Case Tracing, Result Caching, etc.

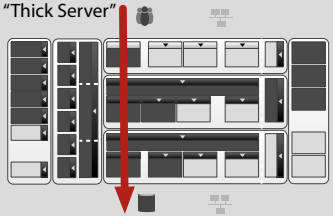
**Data Facade (DF) Potential Functionality:**
Data Access Authorization, Data Orchestration (Dispatching), Data Splitting/Aggregation/Conversion, Transaction Handling, Run-Time Data Access Tracing, Data Caching, etc.

Input/Output Interface
Application Programming Interface (API)

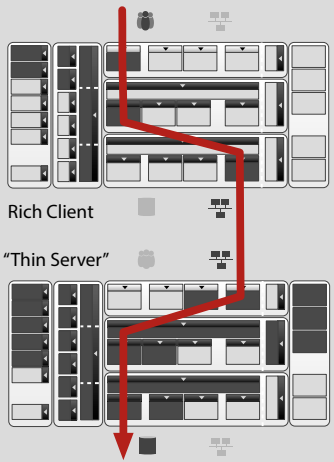
## Blueprint Usage

### Example Scenario 1

Thin Client / "Thick Server"



### Example Scenario 2




Intellectual Content: Version 1.3.5 (2023-07-15) Authored 2007-2023 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.2.3 (2022-10-09) Copyright © 2011-2022 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Reactive System Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

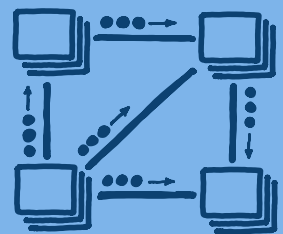
AF 10.4

## Architecture & Systems

### DEF Definition

*Reactive System Architecture* enables the realization of *Reactive Systems*.

Reactive Systems are in *subordinated interaction* with their *dominating environment*. They *continuously* process *endless data streams* as *small messages*, react at *any time* and respond within *tight time limits*. For this, they continuously *observe* their *environment* and adapt their *behaviour* to the current situation.



## Demand & Deliverables

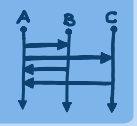
### CTX Context

*Real-time communication* in the context of Digitization, Internet, Internet of Things (IoT), Systems of Engagement, Media and Analytics.



### VAL Values

*Non-blocking input/output data processing, fast responses* within *tight time limits*, and *continuous availability* of the provided services.



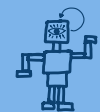

### REQ Requirements

Services are *elastic* and provide high scalability, and are *resilient* and provide high fault tolerance.



### PRP Properties

Services run *fully autonomously, monitor themselves*, and *automatically adapt* to changes in the environment.



## Principles

**Stay Responsive**
Always respond in a timely manner.

**Accept Uncertainty**
Build reliability despite unreliable foundations.

**Embrace Failure**
Expect things to go wrong and design for resilience.

**Assert Autonomy**
Design components that act independently and interact collaboratively.

**Tailor Consistency**
Individualize consistency per component to balance availability and performance.

**Decouple Time**
Process asynchronously to avoid coordination and waiting.

**Decouple Space**
Create flexibility by embracing the network.

**Handle Dynamics**
Continuously adapt to varying demand and resources.

## Patterns & Paradigms

### ARC Architecture

Microservices, Cloud-Native Architecture (CNA), Event-Driven Architecture (EDA).




### COM Communication

Asynchronous Communication, Non-Blocking I/O, Sequence, Push, Backpressure, Quality of Service (QoS).



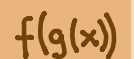
### DAT Data

Semantical Event, Small Message, Endless Stream.



### STY Style

Functional Programming, Asynchronous Programming.




### EXE Execution

Parallelization, Concurrency, Actors, Threads, Thread-Pool, Event-Loop.



### INF Infrastructure

Message Queue (MQ), Load Balancer, Reverse Proxy, Service Mesh, Virtual Private Network (VPN).




### PRC Processing

Complex Event Processing (CEP), EAI Patterns, Stream Processing (map, flatMap, filter, reduce), Event Sourcing.



### ASY Asynchronism

Callback, Promise/Future, Observable, Publish & Subscribe.



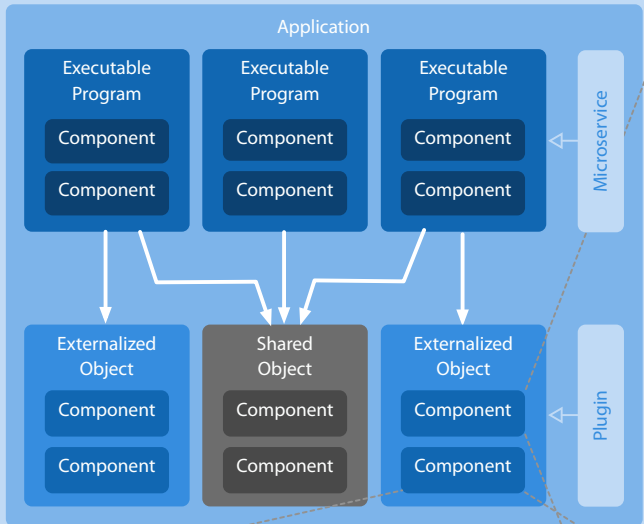

Intellectual Content: Version 1.1.1 (2020-11-05), Authored 2019-2020 by Michael Schäfer and Dr. Ralf S. Engelschall at mag. Applied Technology Research
Graphical Illustration: Version 1.1.0 (2020-09-26), Copyright © 2019-2020 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

# Application Composition

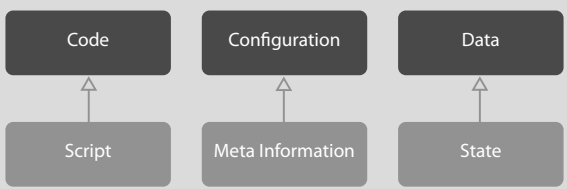




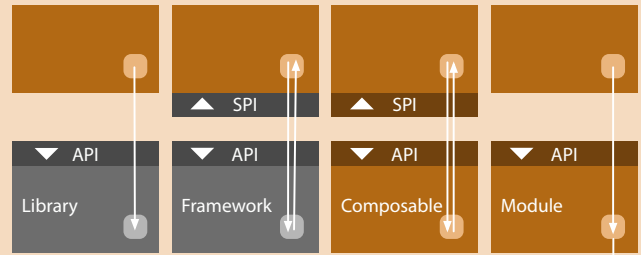
## Application Structure



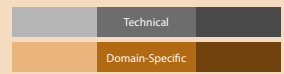

## Component Assets




## Component Types




Legend:




*Applications* are composed out of one or more *Executable Programs*.

An *Executable Program* in turn can leverage zero or more (dedicated) *Externalised Objects* and/or (reusable) *Shared Objects*. Both *Executable Programs*, *Externalised Objects* and *Shared Objects* are composed through one or more *Components*.

*Components* are comprised of the assets *Code*, *Configuration* and *Data* and are of four distinct types: *Library*, *Framework*, *Composable* and *Module*, which are distinguished by their particular combination of characterizing aspects.

| | data persistence | own life-cycle | requires integration | self-contained | invasive | passive | active | technical | domain-specific | reusable |
|---|---|---|---|---|---|---|---|---|---|---|
| Library | | | | X | | X | | X | | X |
| Framework | | X | X | X | X | X | X | X | | X |
| Composable | X | X | X | X | | | X | | X | X |
| Module | X | | | | | | X | | X | |

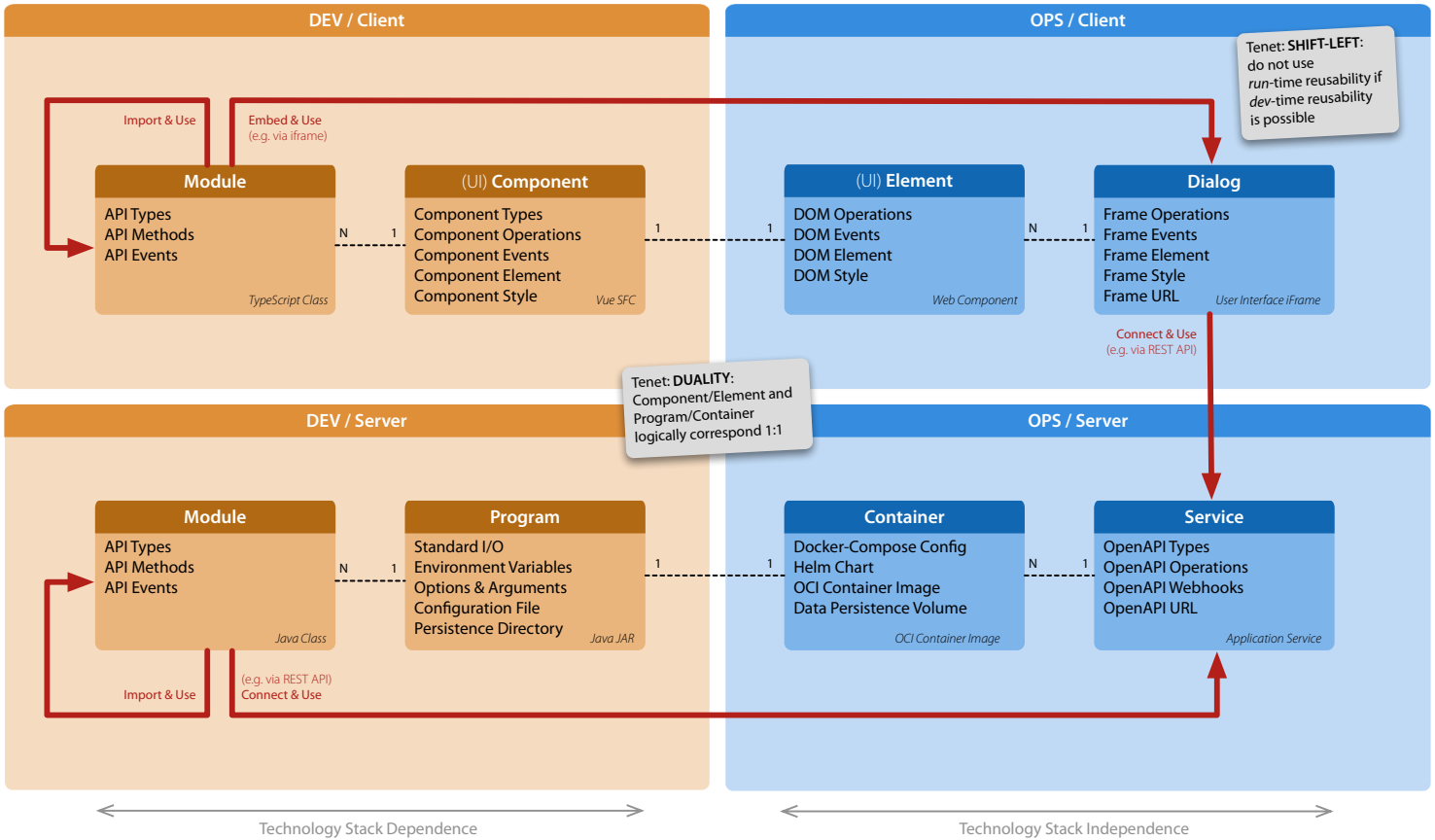
Intellectual Content: Version 1.0.2 (2022-05-24) Authored 2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.2 (2022-05-24) Copyright © 2022 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Functional Reusability

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 11.4




Intellectual Content: Version 1.1.0 (2025-09-15), Authored 2024-2025 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.1.0 (2025-09-15), Copyright © 2024-2025 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Formal Languages

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 12.1

## Declarative Languages

**Express the target state**
and let the machine figure out the steps.

### Markup Languages

Write text intermixed with markup information.



```
foo <em>bar <strong>baz
</strong></em> quux
```

Examples:
**Wiki**, **Markdown**, AsciiDoc, SGML, **HTML**, TeX, R[un]off, reStructuredText, RTF

### Configuration Languages

Express complex textual configurations.



```
foo bar quux { baz;
quux id 7; baz }
```

Examples:
INI, **XML**, SXML, **JSON**, **YAML**, TOML, HCL

### Rule Languages

Express logic and semantic through complex rules.

§

```
foo(x, y) <- bar(x, y, z)
AND x < 42 AND z >= 10
```

Examples:
**SQRL**, Datalog/RuleML, OWL/**SWRL**, RIF

### Constraint Languages

Find solutions for complex constraints.



```
foo @ bar(X, Y),
baz(X, Y, _) ==> quux.
```

Examples:
**MiniZinc**, CHR, **OCL**, Rego, Z3.

### Query Languages

Retrieve information through paths and expressions.



```
// foo / bar [ @baz ==
"xxx" && @quux > 10 ]
```

Examples:
Glob, **RegExp**, **CSS Selector**, **XPath**, YARA, **GraphQL**, **SQL**, SPARQL, Cypher, GQL, **ASTq**

### Validation Languages

Parse and validate complex textual information.



```
foo ::= “bar(#” (?:
[0-9a-fA-F]{2})+ “)”
```

Examples:
**RegExp**, Ducky, **BNF**, **PEG**, RELAX NG

solution approach: automatically, non-obvious
execution control: automatically, pre-defined
performance optimization: automatically, pre-defined

## Imperative Languages

**Express the steps**
how the machine has to reach the target state.

### Shell Languages

Automate execution of system commands.



```
foo -x 2>&1 | bar -y
--quux <(cat *.cf)
```

Examples:
Korn-Shell, Bourne-Shell, **Bash**, C-Shell, Batch-Script, **PowerShell**, AppleScript, DCL

### Programming Languages

Execute complex algorithmic steps.




```
for (let i = 0; i < 10;
i++) foo(i, 42)
```

Examples:
**JavaScript**, **TypeScript**, **Scala**, **Kotlin**, Java, C#, C/C++, **Rust**, **Go**, Python, Perl, Ruby, Lua

### Text-Processing Languages

Manipulate texts through transformations.




```
/^foo/,/bar.*baz/
s/quux\([0-9]*\)/foo\1/g
```

Examples:
ed, ex, **sed**, AWK, TXR, XSLT, JSLT

### Macro Languages

Pre-process texts with macros.



```
define(`foo',`bar$1baz')
foo(quux)bar
```

Examples:
**m4**, GPP, CPP, **Zoem**, ProMac

### Expression Languages

Expand path, arithmetic, and boolean expressions.

f(x)

```
{{ foo.bar[*].baz[42]
.quux + 1 }}
```

Examples:
**JQ**, **YQ**, **MozJEXL**, MathML, JUEL, SpEL

### Template Languages

Expand complex text fragments.




```
{% for k, v in items %}
{{k}}: {{v}}{% endfor %}
```

Examples:
Pug, **Nunjucks**, Handlebars, Mustache, Jinja, **Jsonnet**

solution approach: manually, obvious
execution control: manually, fine-grained
performance optimization: manually, fine-grained

Examples:
**essential**
recommended
alternative

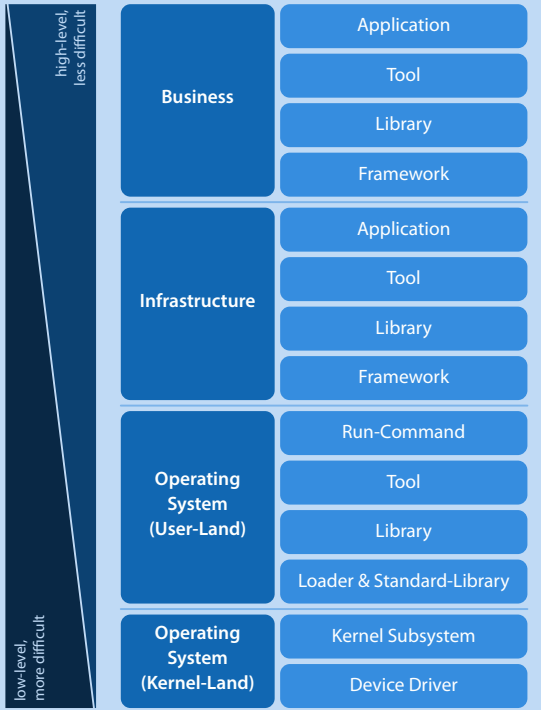
Intellectual Content: Version 1.0.5 (2020-10-17), Authored 2019-2020 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.1 (2021-07-19), Copyright © 2019-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

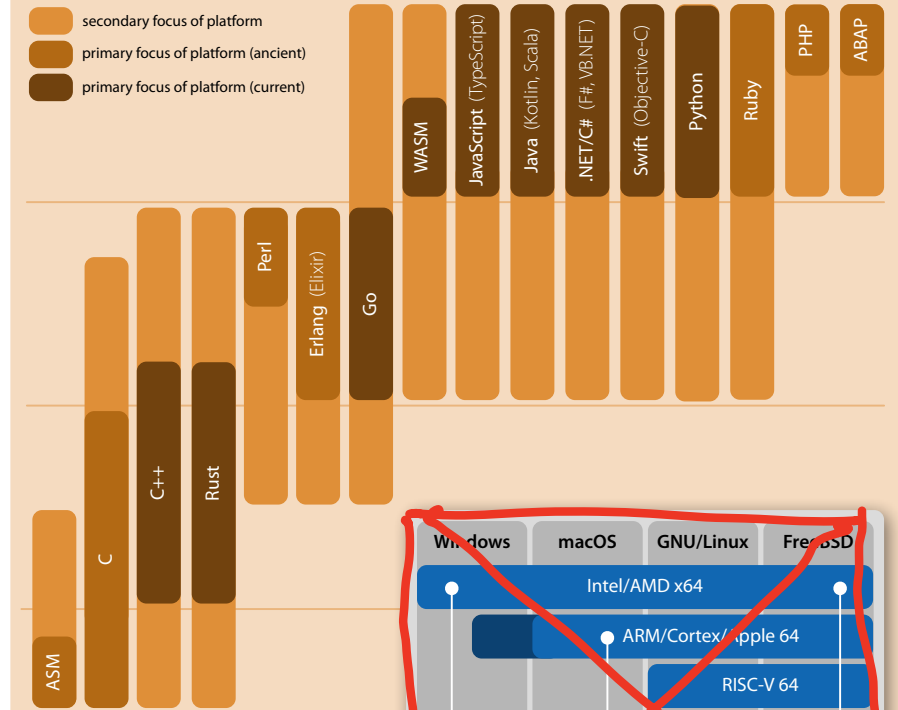
ARCHITECTURE FUNDAMENTALS

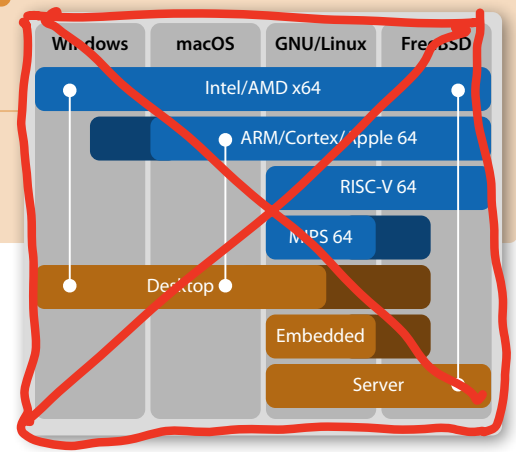
# Technology Platforms

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 12.3




**Remember:**
A *Technology Platform* is less about choosing a particular programming language and more about choosing a particular ecosystem for targeting a particular level of software!

**Opinionated Recommendation** (as of 2022):

| | |
|---|---|
| Business: | **Scala, Kotlin, TypeScript, AssemblyScript** |
| Infrastructure: | **Go, Rust, Scala, Kotlin, TypeScript** |
| Operating System (UL): | **Rust, Go** |
| Operating System (KL): | **C, C++, Rust** |

**Typical Computing Devices** (as of 2022):

| | |
|---|---|
| Intel/AMD x64: | **Personal Computer (PC)** |
| ARM/Cortex/Apple 64: | **Raspberry PI, BeagleBone, ROCKSPro64, iMac** |
| RISC-V 64: | **Beagle-V, HiFive Unmatched** |
| MIPS 64: | **Compex WPJ344** |

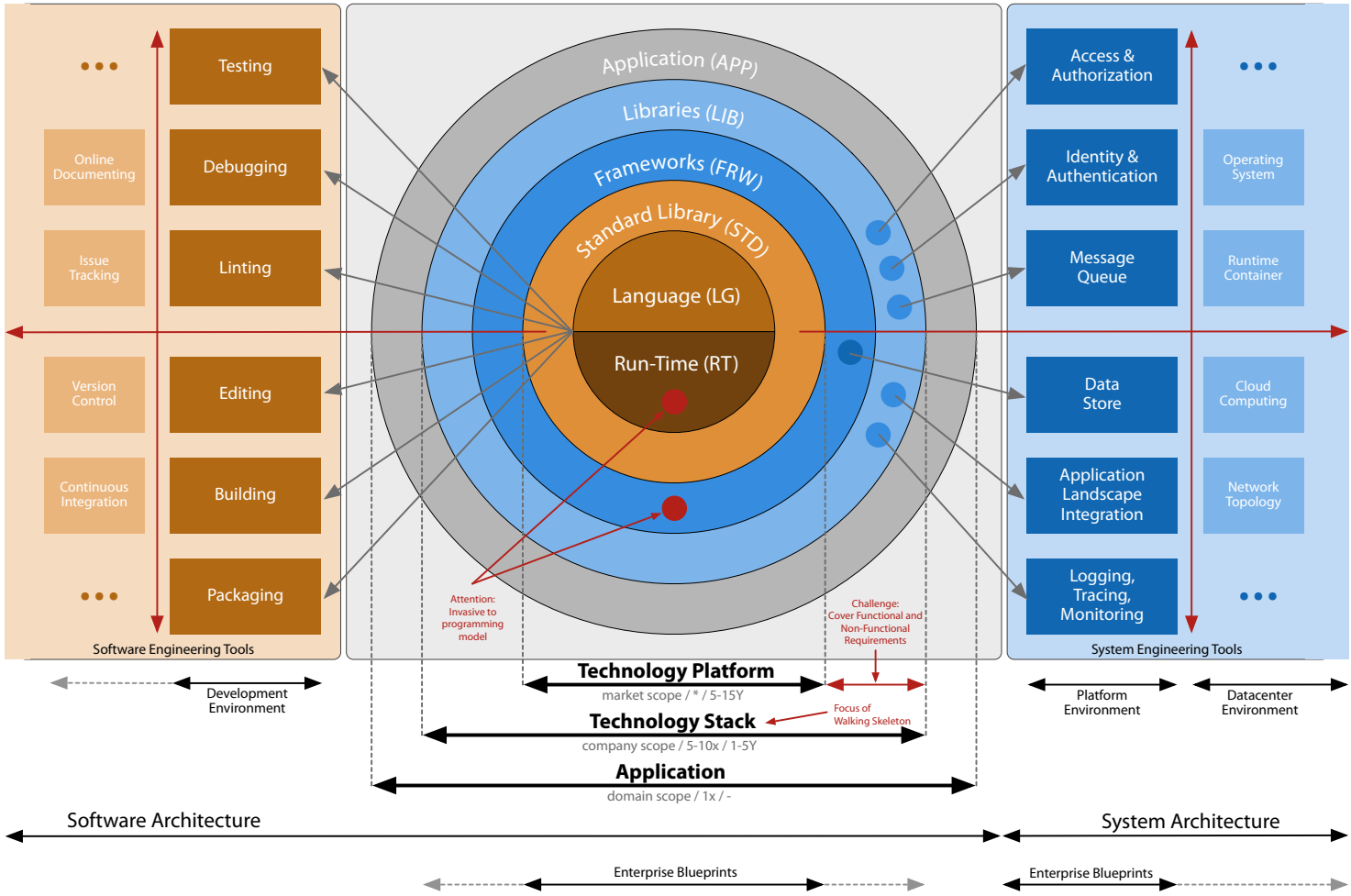
Intellectual Content: Version 1.1.0 (2021-07-23) Authored 2020-2021 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.1.0 (2021-07-23) Copyright © 2020-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Technology Stack

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 12.4




Intellectual Content: Version 1.0.5 (2019-10-27), Authored (2017-2019) by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.6 (2021-10-05), Copyright © 2017-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Rich-Client Aspects

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 12.5

## IT Interface Theme

Style Reset, Shape, Color, Gradient, Shadow, Font, Icon

**Bootstrap** | **TypoPRO, FontAwesome, Normalize**



## IW Interface Widgets

Icon, Label, Text Paragraph, Image, Form, Text-Field, Text-Area, Date Picker, Toggle, Radio Button, Checkbox, Select List, Slider, Progress Bar, Hyperlink, Popup Menu, Dropdown Menu, Toolbar, Tooltip, Tab, Pill, Breadcrumb, Pagination, Badge, Alert, Panel, Modal, Table, Scrollbar, Carousel

**Bootstrap** | Select2, SlickGrid, ...




## IL Interface Layouting

Responsive Design, Media Query, Frame, Grid, Padding, Border, Margin, Alignment, Force, Magnetism

**Bootstrap** | Swiper, jQuery Page, ...



## IE Interface Effects

Transition, Transformation, Keyframes, Easing Function, Sound Effect, Physics

**VueJS** | **Animate.css,** DynamicJS, Howler, ...



## II Interface Interactions

Mouse, Keyboard, Touchscreen, Gesture, Clipboard, Drag & Drop

**VueJS** | **Hammer, Mousetrap,** Dragula, ...



## IS Interface States

Rendered, Enabled, Visible, Focused, Warning, Error, Floating

**VueJS** | (none)




## IM Interface Mask

Markup Loading, Markup Generation, Virtual DOM, Text, Bitmaps, Vectors, 2D/3D Canvas, Accessibility

**VueJS** | **jQuery-Markup,** D3, Snap.svg, FabricJS, ...



## 18 Interface Internationalization

Text Internationalization (I18N).

**VueJS** | vue-i18next, I18Next




## DC Data Conversion

Value Formatting, Value Parsing, Localization (L10N).

**VueJS** | Moment, Numeral, Accounting, ...




## DB Data Binding

Reactive, Observer, Unidirectional, Bidirectional, Incremental

**VueJS** | (none)




## PM Presentation Model

Parameter Value, Command Value, State Value, Data Value, Event Value, Value Validation, Presentation Logic

**ComponentJS** | (none)




## DN Dialog Navigation

Deep Linking, Routing, Dialog Flow

**ComponentJS** | **Director,** URI.js




## DA Dialog Automation

Dialog Macros, Click-Through, Smoke Testing.

**ComponentJS** | **ComponentJS-Testdrive**



## DC Dialog Communication

Service, Event, Model, Socket, Hooks

**ComponentJS** | **Latching**



## DL Dialog Life-Cycle

Component States, Component State Transitions.

**ComponentJS** | (none)



## DS Dialog Structure

Component, Model/View/Controller Roles, Hierarchical Composition

**ComponentJS** | ComponentJS-MVC



## SP State Persistence

Local Storage, Cookies, Caching

**(none)** | Store.js, JS-Cookie




## BM Business Model

Entity, Field, Relationship, Universally Unique Identifiers (UUID)

**(none)** | DataModelJS, Pure-UUID




## UA Use-Case Authorization

User Experience, Dialog Restriction, User, Group, Role, Use-Case, Data, Access.

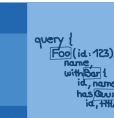
**(none)** | (none)




## CN Client Networking

Request/Response, Synchronization, Push, Pull, Pulled-Push, REST, GraphQL, Authentication, Session.

**(none)** | **Axios,** Apollo Client




## ED Environment Detection

Runtime Detection, Feature Detection.

**(none)** | Modernizr, FeatureJS, jQuery-Stage



Intellectual Content: Version 11.5 (2020-07-24) Authored 2015-2020 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.5 (2020-07-24) Copyright © 2015-2020 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Thin-Server Aspects

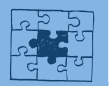
TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 12.6

## ED Environment Detection

Detect the run-time environment, like underlying operating system, execution platform, network topology, feature toggles, etc.

Node | process, syspath



## AP Argument Parsing

Parse options and arguments of the Command-Line Interface (CLI) to bootstrap application parameters.

(none) | yargs



## CP Configuration Parsing

Load and parse directives from configuration file to bootstrap application parameters.

(none) | js-YAML



## PD Process Daemonizing

Detdetach from the startup terminal and host process in order to run fully independently.

(none) | daemonize2



## PM Process Management

(Pre-)fork child processes and/or threads of execution and monitor and control them during the life-cycle of the application.

(none) | cluster, nodemon



## CM Component Management

Structure the code into components, instanciate them under run-time and manage them in a stateful component life-cycle.

Microkernel | (none)



## CC Component Communication

Provide inter-component communication mechanisms like events, hooks, registry, etc.

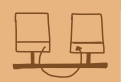
Microkernel | Latching



## SN Server Networking

Listen to network sockets, accept connections and manage request/response and message communication.

HAPI | hapi-plugin-websocket, ws



## PI Peer Information

Determine unique identification and add-on information about the client peer.

HAPI | hapi-plugin-peer, geoip



## SH Session Handling

Manage secured per-connection sessions to keep state between communication requests and/or client sessions.

HAPI | YAR




## UA User Authentication

Determine and validate the unique identity of the user communicating over the current network connection.

HAPI | JWT, Passport



## RV Request Validation

Validate the syntactical and semantical compliance of the requests and sanitize the requests.

HAPI | Joi, DuckyJS




## RP Request Processing

Process the request by dispatching execution according to the provided request and determined context information.

HAPI | GraphQL.js



## RA Role Authorization

Determine whether the role of the current user is allowed to execute the current request.

(none) | GraphQL-Tools-Sequelize



## CN Client Networking

Provide mechanisms to connect to peers over the network and perform request/response and/or publish/subscribe communication.

(none) | Axios, MQTT.js, ws



## TS Task Scheduling

Schedule and execute recurring tasks independent of regular I/O operations.

(none) | node-scheduler



## ET Execution Tracing

Provide mechanisms for tracing the execution by logging event and measurement information at certain points of interest.

Microkernel | Winston



## DA Database Access

Map in-memory domain entities onto data store dependent persistent data structure.

Sequelize | GraphQL-Tools-Sequelize




## DC Database Connectivity

Locally or remotely connect the database access layer to the underlying data store.

Sequelize | sqlite3, pg



## DS Database Schema

Create, update or downgrade the data schema inside the underlying data store.

Sequelize | (none)



## DB Database Bootstrapping

Create, update or downgrade both mandatory bootstrapping and optional domain-specific data inside the underlying data store.

Sequelize | ini



Intellectual Content: Version 1.0.6 (2020-10-17), Authored 2017-2020 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.1 (2018-10-25), Copyright © 2017-2018 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to technische.universitaet.muenchen (TUM) for reproduction in Computer Science lecture context only.

ARCHITECTURE FUNDAMENTALS

# Software Deployment

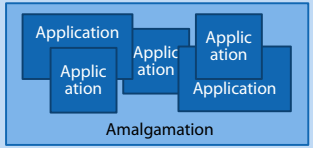
TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 13.1

## AMA Bare Amalgamation

Manually deploy all applications into a single, shared, and unmanaged filesystem location. Dependencies are resolved manually. Examples: Windows Fonts, Unix 1990th /usr/local.

**Pro:** simple deployment
**Con:** incompatibilities, hard uninstallation




## UHP Unmanaged Heap

Manually deploy all applications into multiple, distinct, and unmanaged filesystem locations. Dependencies are resolved manually. Examples: macOS *.app, OpenPKG LSYNC.

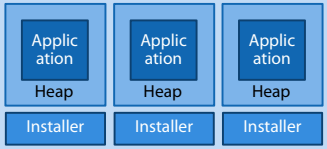
**Pro:** simple deployment, easy uninstallation
**Con:** no repair mechanism




## MHP Managed Heap

Let individual installers deploy applications into multiple, distinct, and managed filesystem locations. Dependencies are manually resolved or bundled. Examples: macOS *.pkg, Windows MSI, InnoSetup.

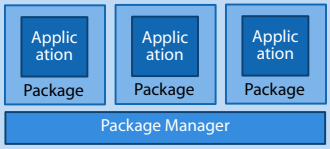
**Pro:** easy uninstallation, repairable
**Con:** requires installer, diversity, no dep.




## PKG Managed Package

Let a central package manager deploy all applications into a single, shared, and managed filesystem location. Dependencies are automatically resolved. Examples: APT, RPM, FreeBSD pkg, MacPorts, Gradle, NPM.

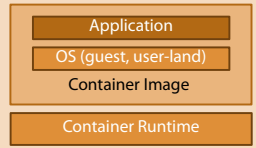
**Pro:** easy uninstall., repairable, dependencies
**Con:** P.M. pre-installation, P.M. single instance




## CON Container Image

Bundle an application with its stripped-down OS dependencies and run-time environment into a container image. Examples: Docker/ ContainerD, Kubernetes/CRI-O, Windows Portable Apps.

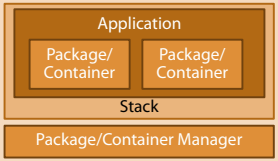
**Pro:** independent, simple deployment
**Con:** fewer variations, no dependencies




## STK Package/Container Stack

Establish an application out of multiple Managed Packages. Examples: OpenPKG Stack, Docker Compose, Kubernetes/ Kompose, Kubernetes/Helm.

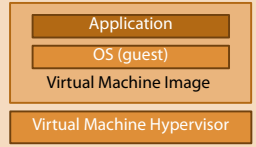
**Pro:** independent, flexible
**Con:** overhead




## VMI Virtual Machine Image

Bundle an application with its full OS dependencies and run-time environment into a virtual machine image and deploy and execute this on a hypervisor. Examples: VirtualBox, VMWare, HyperV, Parallels, QEMU.

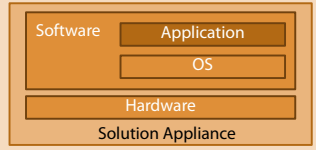
**Pro:** all-in-one, independent
**Con:** overhead, sealed, inflexible




## APP Solution Appliance

Bundle an application with its full OS dependencies, run-time environment and underlying hardware. Examples: AVM Fritz! Box, SAP HANA.

**Pro:** all-in-one, independent
**Con:** expensive, sealed, inflexible




Intellectual Content: Version 1.0.1.0 (2022-07-01), Authored 2018-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.7 (2019-11-03), Copyright © 2018-2019 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Cloud Computing Resources

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 13.2

## Cloud Characteristics — How?

logically / chronologically related

- ON-DEMAND
- SELF-SERVICE
- AUTOMATABLE
- NETWORK ACCESS

logically / chronologically related

- MULTI-TENANCY
- MEASURED SERVICE
- ELASTICITY
- PAY-PER-USE

## Cloud Scope — Who?

outside-inside relationship

- Public Cloud
- Community Cloud
- Private Cloud

## Cloud Location — Where?

outside-inside relationship

- Provider-Hosted
- On-Premises

## Cloud Approaches — What?

technical layers

- Service
- Application
- Service Mesh
- Service Events
- Application Runtime
- Tools & Libraries
- Container Runtime
- Operating System
- Computer
- Network

| | SaaS | FaaS | PaaS | CaaS | IaaS |
|---|---|---|---|---|---|
| | e.g. Salesforce<br>e.g. Sharepoint 365<br>(think: Web-UI-based Service) | e.g. AWS Lambda<br>e.g. Azure Functions<br>(think: Stateless Microservice) | e.g. AWS EKS<br>e.g. Azure AKS<br>(think: Kubernetes) | e.g. AWS ECS & Fargate<br>e.g. Azure Container Apps<br>(think: Docker) | e.g. AWS EC2<br>e.g. Azure Virtual Machines<br>(think: Virtual Machine) |
| | | aka "Serverless Computing" | | | |
| | **Software** as a Service | **Function** as a Service | **Platform** as a Service | **Container** as a Service | **Infrastructure** as a Service |
| Duration: | User Session | Function Call | Continuously | | |
| Interfaces: | Proprietary | | | Standardized | |
| | | | | (think: OCI) | (think: PC) |

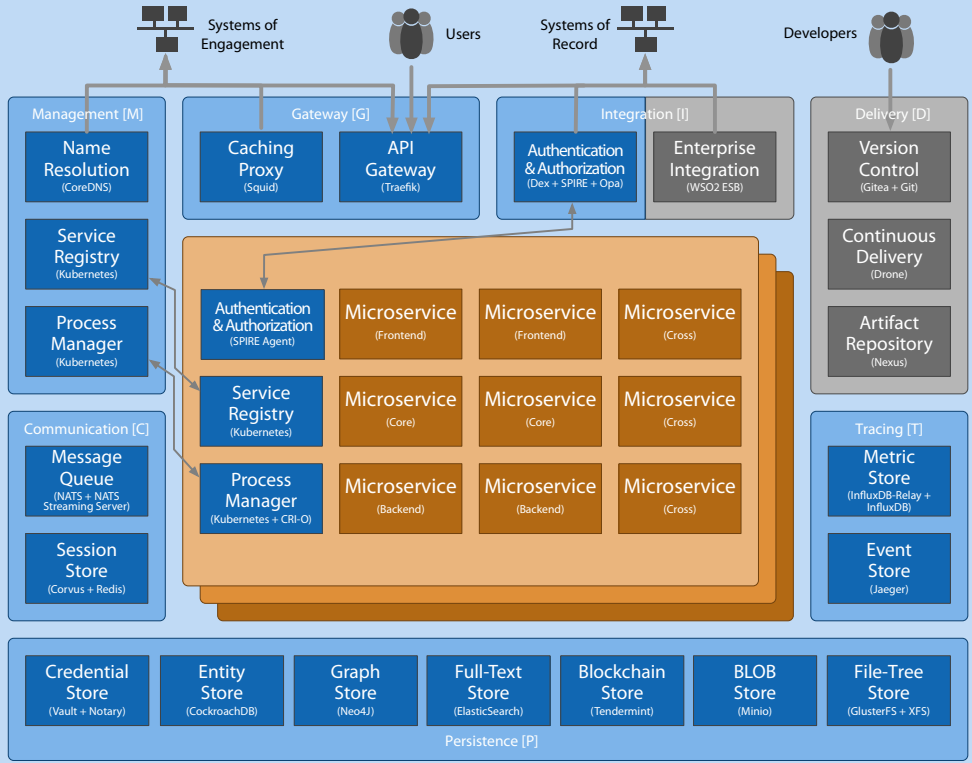
Intellectual Content: Version 1.20 (2022-07-01) Authored 2017-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.30 (2022-07-01) Copyright © 2017-2022 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Cloud-Native Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

## Reference Architecture Blueprint




Common Group
Microservice Group
(Virtual) Machine
Platform Application (HA)
Foreign Application (non-HA)
Microservice Application (HA)

**Major Approach Idea:**
With Cloud-Native Architecture one maximizes the leverage of PaaS-like, high-available, and scalable Cloud services at the level of Software- and Systems-Architecture for a whole set of applications.

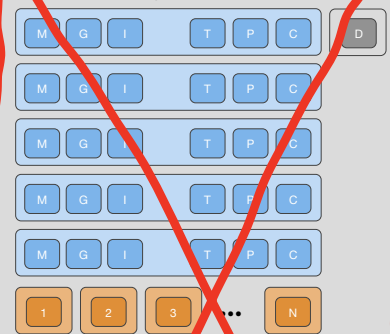
**Major Design Criterias:**
1. Targets DevOps approach.
2. Targets Continuous Delivery process.
3. Targets Microservice Architecture.
4. Targets Container Image deployment.
5. Targets Service Mesh communication.
6. Targets Server Cluster setup.
7. Provides High-Availability of Service Platform
8. Provides High-Availability of Application Microservices.
9. Provides Scalability of Application Microservices.

## CNCF Cloud-Native Definition 1.0

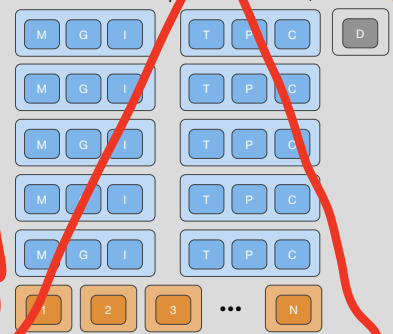
Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds; containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

## Practical Cluster Setups

Standard Cluster Setup (5+1+N Machines):




Partitioned Cluster Setup (3x2+1+N Machines):




AF 13.3

Intellectual Content: Version 1.24 (2019-10-26), Authored 2016-2019 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.2.6 (2021-10-09), Copyright © 2016-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Offline Capability

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 13.4

## GOAL

### UX Exceptional User Experience

Provide exceptional *User Experience* by taking temporary client/server network offline situations into account and establish trust in the offline capability of the application.

## CONTEXT

### IS Business Information Systems

Client/server applications which drive business processes through use-cases, based primarily on the editing, storing, and retrieving of information.

### CC Cloud Computing

On-demand availability of computing resources, especially data storage and computing power, without direct active management by the customer.

## CHALLENGES

### VPN Toggled Virtual (Private) Networks

Network offline situations caused by the explicit on/off toggling of overlayed *Virtual (Private) Networks* by the user.

### MNC Switched Mobile Network Cells

Network offline situations caused by the implicit switching between the mobile network cells by the device during mobile use.

### NCO Failed Network Components

Network offline situations caused by the failure of any network components between the client and server tiers of an application.

## MATURITY LEVELS

**Hint:** you cannot solve offline scenarios at the technical level if your use-cases are not already aligned to them.

### L2 Offline Read

During offline phase, client allows **read**-operations, but **no write**-operations, to locally cached data.

*Data Access Layer Proxy*
*Web Storage API*

### L1 Offline Aware

During offline phase, client **explicitly** disables user interface and **shows modal error message**.

*Rich-Client*
*Web Network Information API*

### L0 Offline Unaware

During offline phase, client **implicitly** fails with network errors.

*Thin-Client*

**Notice:** each level increases user experience, but also increases technical complexity.

### L5 Offline Transactional Read / Write

During offline phase, client allows **non-atomic (transactional)** read/write-operations to any locally cached data.

*SAGA Compensation*

### L4 Offline Read & Atomic Write

During offline phase, client allows **atomic read/write**-operations to **any** locally cached data.

*CRDT, Event Sourcing*
*CQRS, Optimistic Locking*

### L3 Offline Read & User-Exclusive Write

During offline phase, client allows atomic **read**-operations to **any**, and **write**-operations to **user-exclusive**, locally cached data.

*Data Synchronisation*
*Last-Win Conflict Resolution*

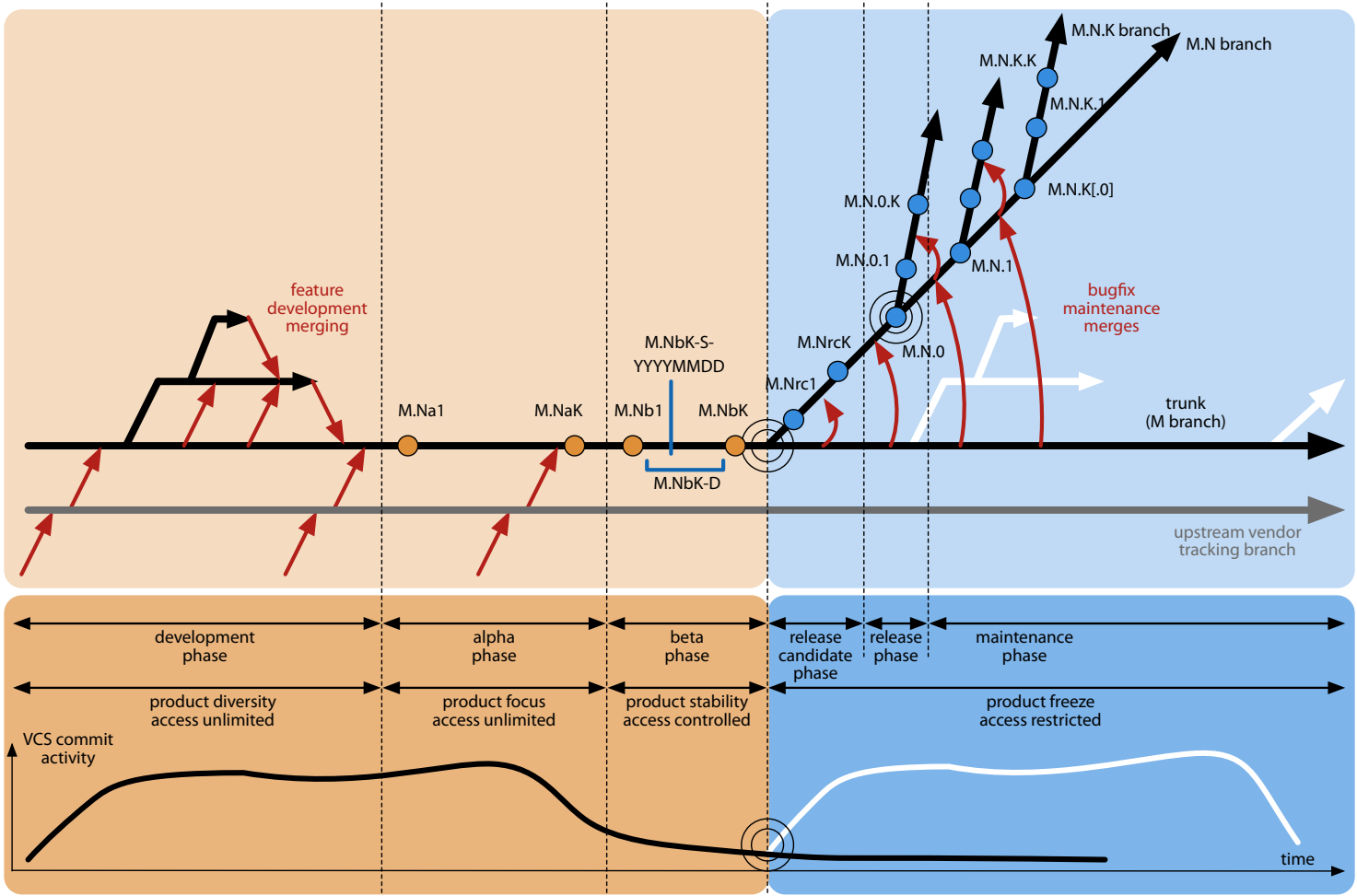
Intellectual Content: Version 1.0.1 (2025-11-06), Authored 2023-2025 by Dr. Ralf S. Engelschall, based on work of Michael Schäfer and Andreas Lehmann
Graphical Illustration: Version 1.0.1 (2025-11-06), Copyright © 2023-2025 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Version Control Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 14.1



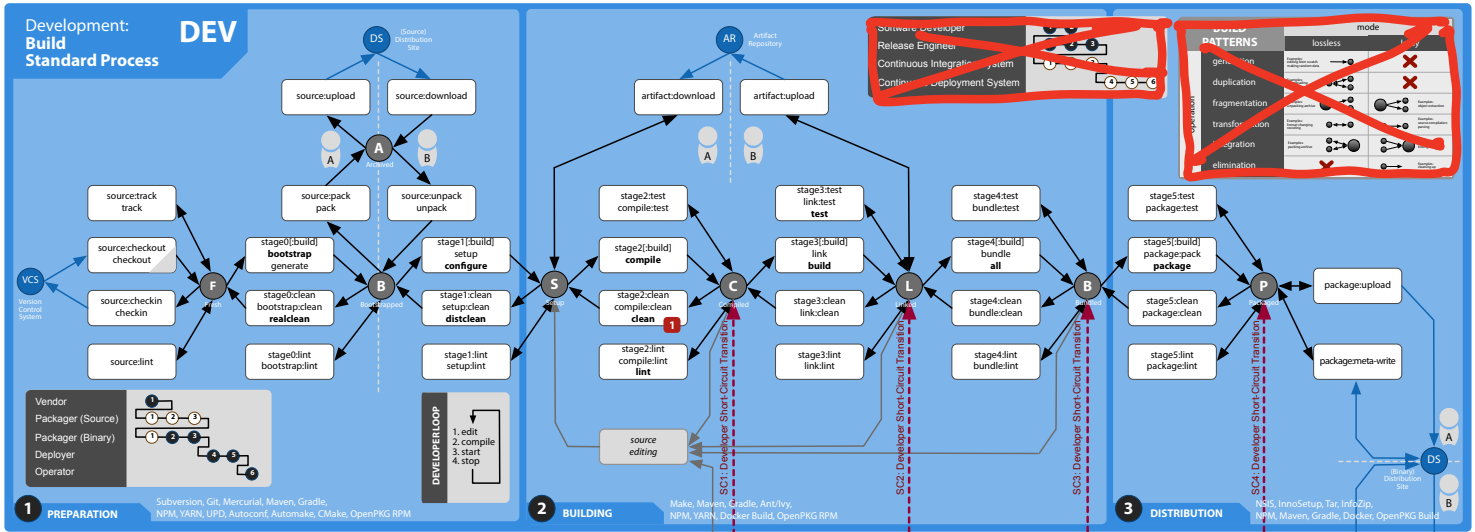

Intellectual Content: Version 1.3.0 (2012-09-19) Authored 2011-2012 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.2.1 (2012-09-19) Copyright © 2008-2012 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Assembly Process Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 14.2




External State/Resource
Process State
State Transition
State Transition (short-circuit)
State Transition (external)
XXX Process Activity (semi-automated or automated)
XXX Process Activity (manually or semi-automated)

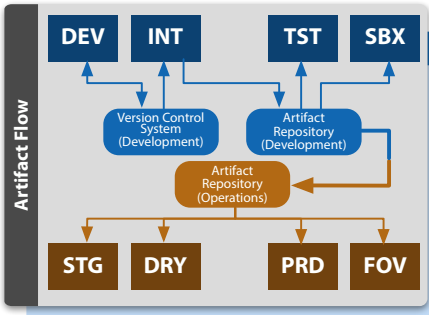
Intellectual Content: Version 13.3 (2020-12-19), Authored 2009-2020 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 13.2 (2013-11-26), Copyright © 2009-2013 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Environments & Quality Assurance

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 14.3




## Development Environments

**DEV Development**
(Potentially destructive) Environment for the (separated) development of all artifacts of the system, typically located on each developer's respective system.

**INT Integration**
(Potentially destructive) Environment for integrating all artifacts of the system, usually located centrally to the developers and driven automatically.

**TST Test**
Environment for testing the system as a whole in a production-resembling context.

**SBX Sandbox**
Destructive environment mirroring the Test environment for demonstration and experimentation purposes, including trainings and showcases.

## Operations Environments

**STG Staging**
Environment for staging the system as a whole in a production-equal context in order to deploy it to the Production environment subsequently.

**PRD Production**
Environment for running the system as a whole in production.

**DRY Dry-Run**
Destructive environment mirroring the Staging environment for experimentation purposes, including operating system and major dependency upgrades.

**FOV Failover**
Destructive environment mirroring the Production environment for failover situation, including disaster recovery situations.

Environment Mirroring

Replicated Environments

Original Environments

Testing Targets:

## Testing Approaches

**UT Unit Testing**
Whitebox testing the inner details of individual units (components) against their functional specification.
Specialization: Regression Testing

**IT Integration Testing**
Whitebox testing the outer interplay of individual units (components) against the technical design of the solution.
Specialization: Smoke Testing

**ST System Testing**
Blackbox testing the system as a whole against the functional and non-functional requirements of the solution.
Specialization: Load Testing

**UAT (User) Acceptance Testing**
Blackbox testing and formal approval of the system as a whole against the end-to-end functionality and user experience requirements.

**OT Operation Testing**
Blackbox testing the system as a whole against the availability and proper operation of the intended services.
Specialization: Service Monitoring

Quality Promise, Bugfix Effort

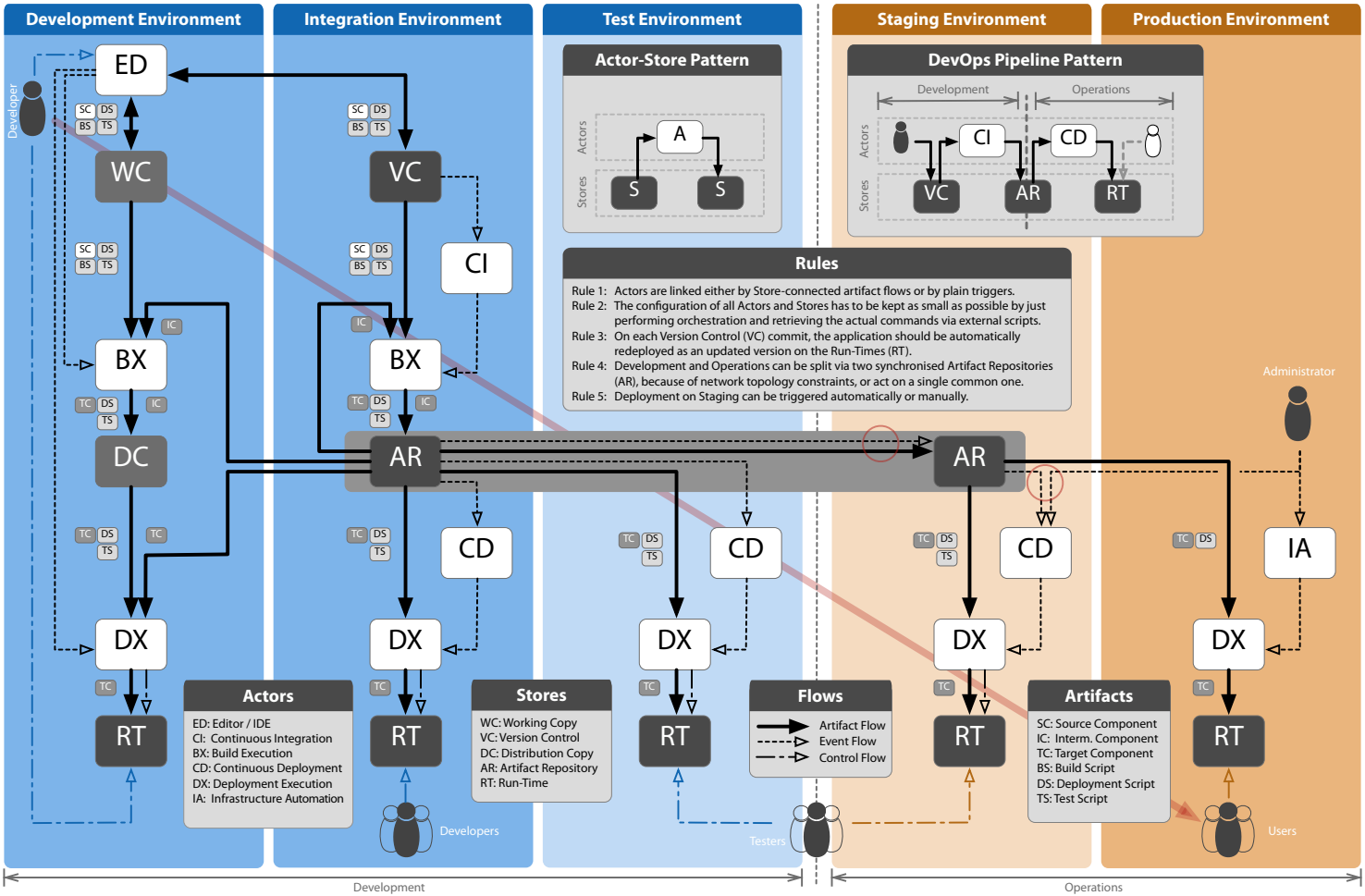
Intellectual Content Version 1.0.6 (2023-11-26), Authored 2018-2023 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.0.4 (2019-09-15), Copyright © 2018-2019 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# DevOps Toolchain

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 14.4




Intellectual Content: Version 1.0.2 (2020-12-19), Authored 2019-2020 by Dr. Ralf S. Engelschall with input from Christian Reibe and Dr. Thomas Schöpf
Graphical Illustration: Version 1.0.2 (2020-10-11), Copyright © 2019-2020 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Software Release Management

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 15.2

## Evolution Stage — What?

**WF** Wireframe
Distraction-free low-fidelity illustration of the solution and its base features, displaying its pure structure and core functionality only.

**WS** Walking Skeleton
Realization of all technical, fundamental aspects of the solution, ensuring the domain specific aspects can be realized later on top of it without problems.

**PT** Prototype
High-fidelity mostly interactive sample, mockup, model or simulation of the solution and its base features, showcasing its structure and functionality.

**MVP** Minimum-Viable Product
Early version of solution with just enough functionality to enable full turn of Build-Measure-Learn loop with minimal amount of effort and time.

**PoC** Proof of Concept
Pure realization of most-risky aspects of the solution, proofing their feasibilities. Might still be based on a different technology than WS, MVP and FP.

**FP** Full Product
Final version of the solution with all intended functionality and targeting the mainstream market.

arbitrary technology ⟷ target technology

## Version Number — When?

**M** Major Version
Major Version of solution. Usually bumped on major technical or domain-specific changes only. A bump resets the Minor Version and the Revision, too.

**N** Minor Version
Minor Version of the solution within the Major Version. Usually bumped on new features. A bump resets the Revision, too.

**R** Revision
The Revision of the Release Phase within Major and Minor Version. Bumped for every A/B/C/R Release Phase.

## Release Phase (p) — When?

**A** Alpha
Early version of the solution with incomplete and unstable functionalities to get feedback on product. Usually tagged as "M.NaR" (R > 0).

**C** Candidate
Mature version of solution with complete and stable functionalities to catch last-minute problems. Usually tagged as "M.NrcR" (R > 0) around RTM.

**B** Beta
Early version of the solution with complete but still unstable functionalities to stabilise product. Usually tagged as "M.NbR" (R > 0).

**R** Release
Release version of the solution with complete and stable functionalities, available for production use. Usually tagged as "M.N.R" (R >= 0).

## Points-In-Time (PiT) — When?

**DEV** Development
Arbitrary permanent points-in-time during development. This is the default tag for the source code. Intended for no availability releases.

**SNP** Snapshot
Distinct temporary point-in-time for a release of the current version without a version increase. Intended for limited availability releases.

**REL** Release
Distinct temporary point-in-time for a release of the current version with a version increase. Intended for early and general availability releases.

## Product Edition — Which?

**CE** Community Edition
Edition of the solution for the Open Source Community. Contains just the base functionality and has limited volunteering support.

**STD** Standard Edition
Edition of the solution with just the standard functionality and regular support.

**EE** Enterprise Edition
Edition of the solution for the Enterprise market. Contains the base and additional functionality and has full commercial support.

**PRO** Professional Edition
Edition of the solution with both the standard and extra functionalities and extended support.

## Availability Scope (S) — Who?

**XA** No Availability
No public availability of solution at all. The scope for all Development and sometimes Snapshot point-in-times.

**EA** Early Availability
Early public availability of solution for early market. Usually for Beta or Release Candidate levels or for Release and initial Release Update levels.

**LA** Limited Availability
Limited public availability of solution. Usually for releases after the End-of-Life-Announcement (EOLA) or for releases with specific customer features.

**GA** General Availability
Late public availability of solution for mainstream market. Usually for Release and sometimes just for Release Update levels.

## Distribution Channel — Where?

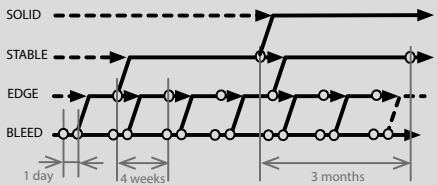
**BLEED** Bleed Channel
Distribution channel for all daily snapshots ("YYYY.MM.DD") with experimental features turned on. Intended for testing purposes only.

**STABLE** Stable Channel
Distribution channel for all quarterly releases ("YYYY.QN") with experimental features turned off. Intended for fast mainstream market and production use.

**EDGE** Edge Channel
Distribution channel for all monthly releases ("YYYY.MM") with experimental features turned on. Intended for early market or testing purposes.

**SOLID** Solid Channel
Distribution channel for all (half-)year releases ("YYYY.N[.N]") with experimental features turned off. Intended for slow mainstream market and production use.

## Distribution Channel Artifacts



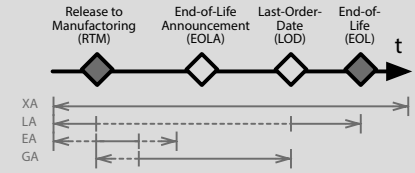

## Versioning Scheme (stdver.org)

M . N p R [ . D ][ + H ][ - S ]

Examples:

```
1.2a3.20230101+F42A-XA
1.2b3-EA
1.2.3-GA
1.2.3
```

## Product Life-Cycle



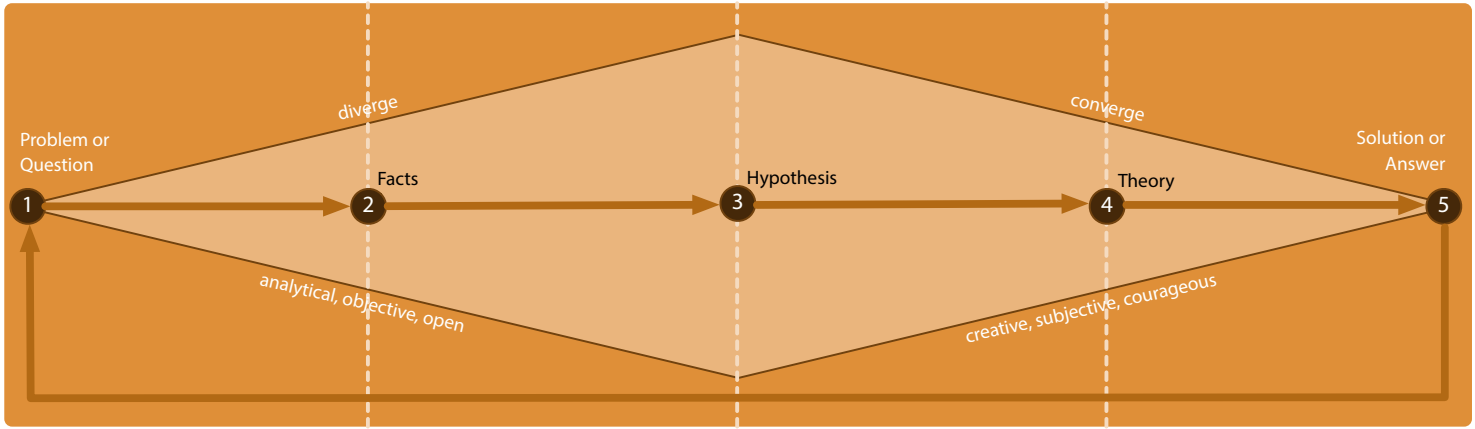

Intellectual Content: Version 1.0.8 (2019-11-16), Authored 2018-2019 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.1.7 (2023-11-26), Copyright © 2018-2023 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Think Clearly

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 16.1




## INVESTIGATE & RESEARCH

1. Reflecting
(find facts via own knowledge/experience)

2. Searching
(find facts via body of knowledge)

3. Verification
(cross-check facts according to sources)

4. Tagging
(classify facts with tags)

## STRUCTURE & SORT

1. Typing
(split/aggregate facts according to type)

2. Clustering
(hierarchically group facts by tags)

3. Relating
(link source to target facts)

4. Ordering
(order facts in each cluster)

## REDUCE & COMPLEMENT

1. Substituting
(substitute/rename facts)

2. Extending
(add still missing facts)

3. Priorization
(priorize facts according to criterias)

4. Rejecting
(reject non-relevant/redundant facts)

## INTEGRATE & PRESENT

1. Specialization
(specialize too general facts)

2. Generalization
(generalise too specific facts)

3. Integration
(aggregate/link facts)

4. Presentation
(convert facts into target form)

(primary step) (secondary step)

Intellectual Content: Version 1.0.1 (2019-06-09), Authored 2010 by Dr. Ralf S. Engelschall, based on work of Mark Lutkowitz.
Graphical Illustration: Version 1.0.2 (2024-11-10), Copyright © 2019-2024 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized reproduction prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Problem Solving Heuristics

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 16.2

## 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.




## 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.

Intellectual Content Version 1.03 (2019-10-31), Authored 2013-2019 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.04 (2021-07-21), Copyright 2013-2021 Dr. Ralf S. Engelschall <http://spdx.org/licenses/CC-BY-NC-ND-4.0> All Rights Reserved
Unauthorized Reproduction Prohibited **Licensed to Technische Universität München for reproduction in Computer Science lecture contexts only.**

ARCHITECTURE FUNDAMENTALS

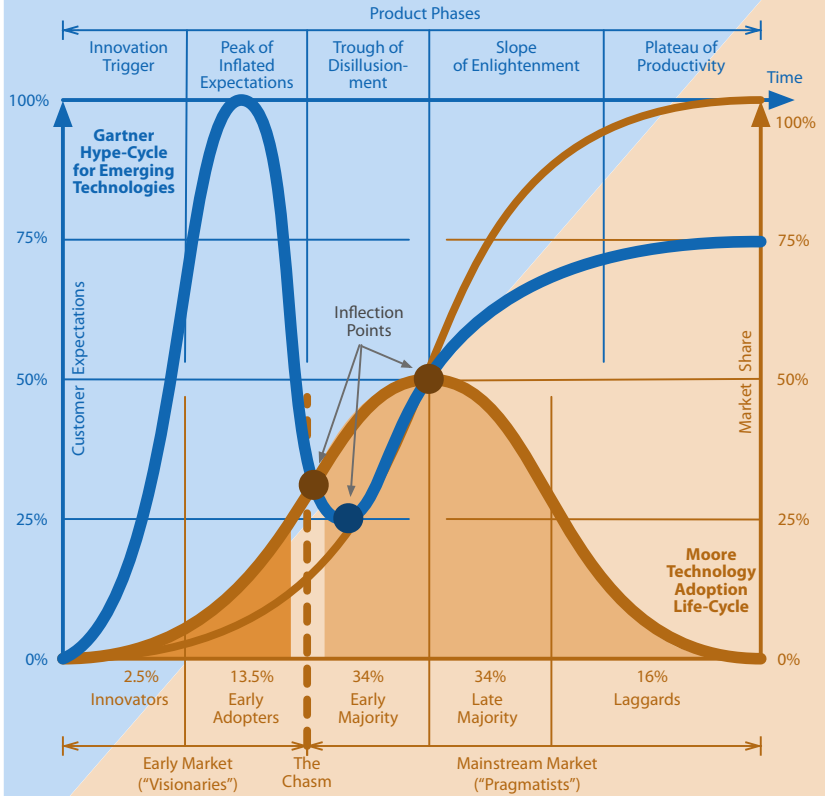
# Technology Life-Cycles

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 16.3

Intellectual Content: Version 1.0.3 (2020-05-16), Authored 2018-2020 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.0.1 (2019-10-31), Copyright © 2018-2019 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited, Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture courses only.

## Gartner Hype-Cycle for Emerging Technologies




## 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 peek 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

ARCHITECTURE FUNDAMENTALS

# Open Source Software

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 16.4

Intellectual Content, Version 1.0.2 (20220404-2), Authored 2007-2022 by Dr. Ralf S. Engelschall
Graphical Illustration, Version 1.0.2 (2022-10-05), Copyright © 2014-2022 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München for reproduction in Computer Science lecture contexts only.

## 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

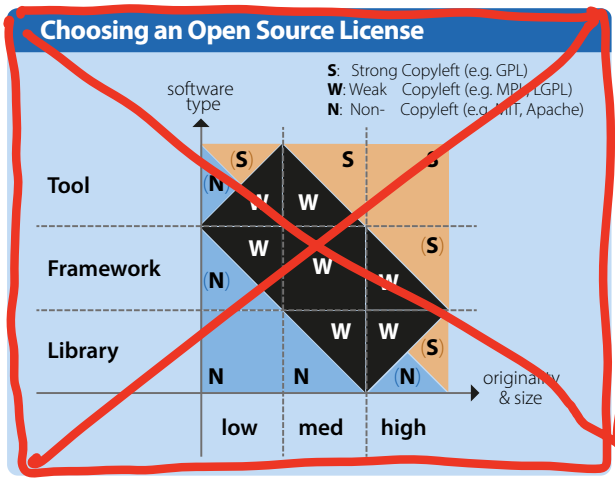
€ **Software Engineering** — Pragmatism, Business, Engineering

Science | Private | Industry

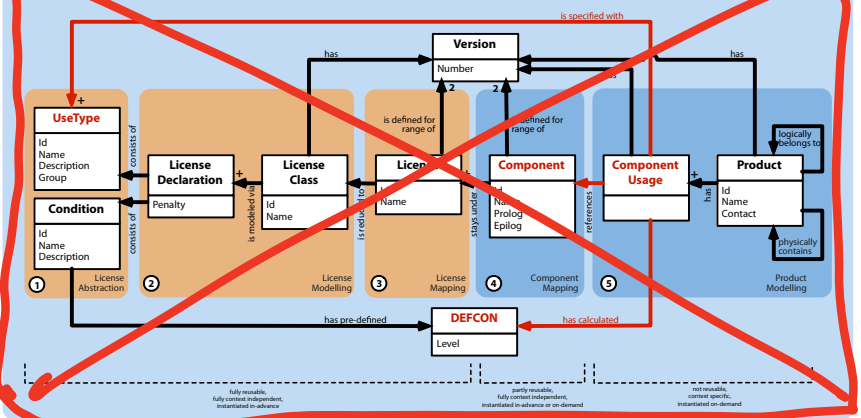
## Most Popular Open Source Licenses

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




## License Compliance Checking Meta-Model



ARCHITECTURE FUNDAMENTALS

# Back of the Envelope Calculation

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 16.5

Intellectual Content Version 1.1.0 (2021-11-23) Authored 2011-2021 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.3.0 (2021-10-10) Copyright © 2012-2021 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture courses only.

## 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

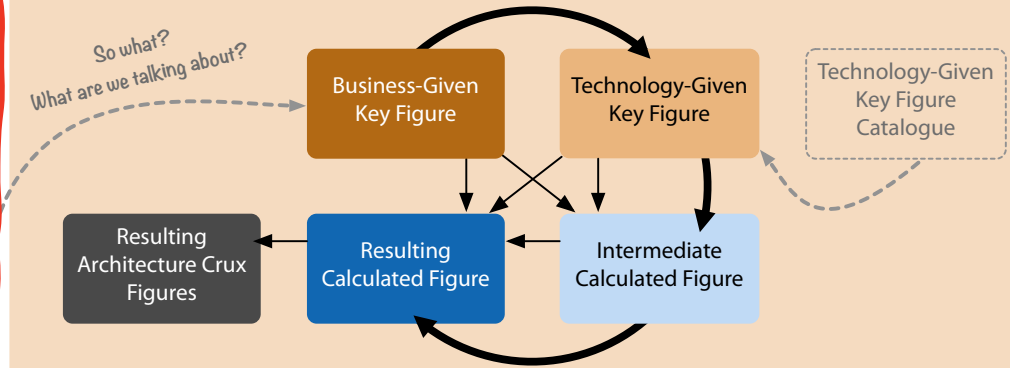
**Use Cases 3/3 (query):**
- user can view timeline (chronological tweets of others he follows)
- user can search for tweets (by keyword "foo", tag "#foo", or user "@foo")
- user can view tag cloud

**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




## 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 Hardware Requirements

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

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

### 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 |

ARCHITECTURE FUNDAMENTALS

# Weighted Decision Matrix

TECHNISCHE UNIVERSITÄT MÜNCHEN

## 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 $A_{best}$ | | | | | | | | | |

## 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.

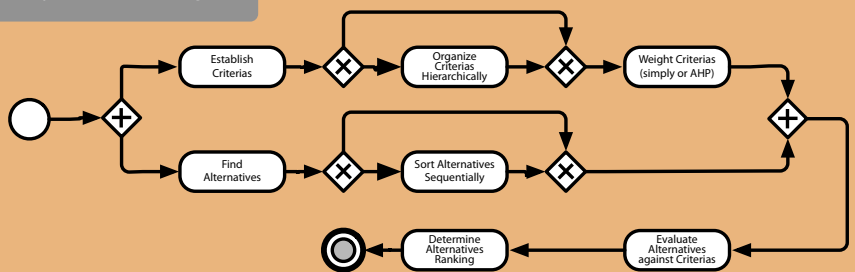
Rule 3: the **best** rating should cover at least **80%** of the **requirements**.

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

## Notice

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

## Decision Making Process




## 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)

AF 16.6

Intellectual Content: Version 1.1 (2022-06-20), Authored 2018-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0 (2021-09-26), Copyright © 2018-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized reproduction, adaptation, distribution or commercial use is prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

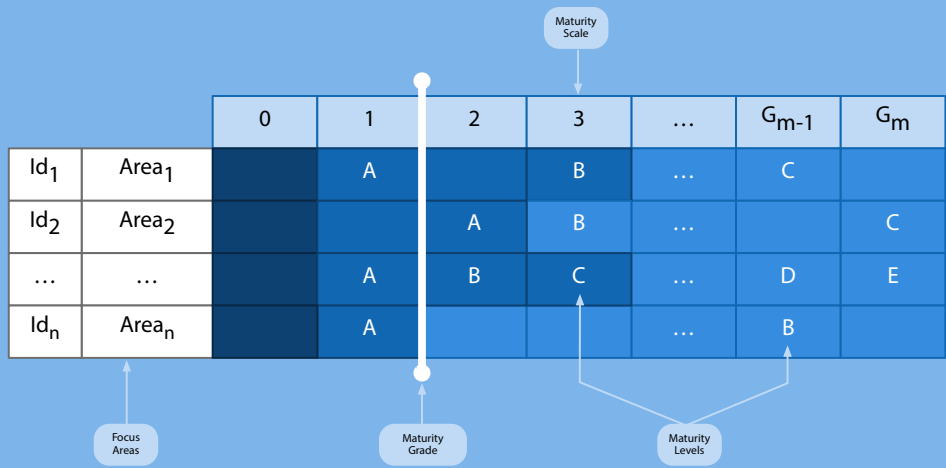
ARCHITECTURE FUNDAMENTALS

# Focus Area Maturity Model

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 16.7

## Matrix **Structure**




## **Focus Area** Definition

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

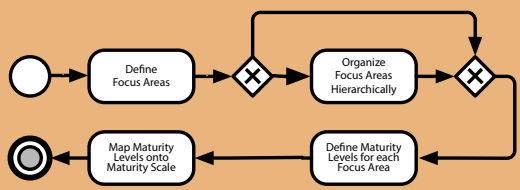
## **Maturity Level** Definition

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>

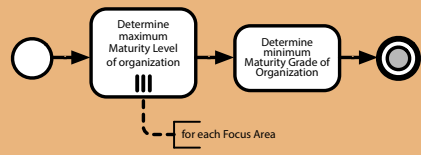
### Maturity Level **Prerequisites**

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




## Maturity **Decision** Process




### 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.

Intellectual Content Version 1.0.1 (2023-12-01) Authored 2022-2023 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.0.0 (2022-05-31) Copyright © 2022 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Big Picture (8-D)

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 17.1

Intellectual Content: Version 1.0.3 (2020-07-20), Authored 2019-2020 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.3 (2020-07-20), Copyright © 2019-2020 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to TECHNISCHE UNIVERSITÄT MÜNCHEN for reproduction in Computer Science lecture courses only.

*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.

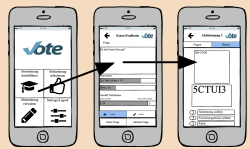
Format: 2xN Table or UML UC Diag.

## 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.

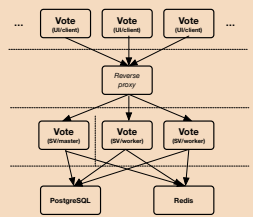
Format: 2xN Table

## 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.

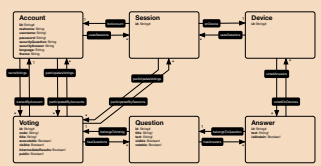
Format: 5xN Table

## 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

ARCHITECTURE FUNDAMENTALS

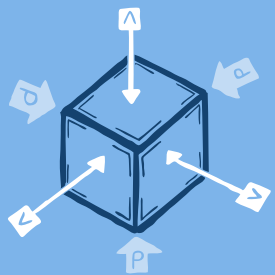
# Viewpoints & Perspectives

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 17.2

## Development (5)

Aspects of the software development process for versioning, building, testing, maintaining, and enhancing the system.



Development Process Viewpoints

## Functionality (2)

System's functional elements, their responsibilities, interfaces, and primary interactions (control flow).

## Information (3)

Static data structures and information flows to store, manipulate, manage, and distribute information.

## Concurrency (4)

Concurrency structure of the system and mapping of functional elements to concurrency units (processes, threads, transaction scopes).

Application Architecture Viewpoints

## Context (1)

Relationships, dependencies, and interactions between the system and its run-time environment (people, systems, external entities).

Application Overview Viewpoints

## Deployment (6)

Required technical environment and mapping of software elements to runtime environment that will execute them.

## Operation (7)

Aspects to operate, administer, update, upgrade and support the system when running in its production environment.

Operation Process Viewpoints

### Evolution & Change

Ability of the system to be flexible in the face of the inevitable change that all systems experience over time.

### Regulation & Compliance

Ability of the system to conform to local and international laws, quasi-legal regulations, company policies, and other rules and standards.

### Internationalization & Localization

Ability of the system to be independent from and adaptable to any particular language, country, or cultural group.

### Usability & Accessibility

Ability of the system to allow people to effectively interact with the system and also to be even used by people with disabilities.

### Performance & Scalability

Ability of the system to predictably execute within its mandated performance profile and to handle increased processing volumes.

### Availability & Resilience

Ability of the system to be fully or partly operational when required and to effectively handle failures.

### Constraints & Resources

Ability of the system to be designed, built, deployed, and operated within known constraints around people, budget, time, and materials.

### Security & Recoverability

Ability of the system to reliably control and audit who can perform what actions on what resources and to detect and recover from failures.

Cross-Cutting Perspectives

Intellectual Content: Version 1.04 (2019-09-15), Authored 2011-2019 by Dr. Ralf S. Engelschall, based on Nick Rozanski, Eoin Woods, "Software Systems Architecture", 2nd ed. 2011, ISBN 978-0-321-71833-4
Graphical Illustration: Version 1.08 (2018-09-15), Copyright © 2011-2018 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Security Basics

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 18.1

## Information Protection Requirements

**Confidentiality**

Information and systems are available to authorized users only.

**Vertraulichkeit** (germ.)

Informationen und Systeme stehen nur autorisierten Benutzern zur Verfügung.



**Integrity**

Correctness of information and functionality of systems is given.

**Integrität** (germ.)

Die Korrektheit der Informationen und die Funktionsweise von Systemen ist gegeben.



**Availability**

Information and systems are available to authorized users without hindrance.

**Verfügbarkeit** (germ.)

Informationen und Systeme stehen autorisierten Benutzen ungehindert zur Verfügung.



## Information Protection Assessment

| | Confidentiality | | | Integrity & Availability |
|---|---|---|---|---|
| | Military: | Gov. (germ.): | Industry: | Recoverable within… |
| very high | top secret | streng geheim | (secret) | 4 hours |
| high | secret | geheim | (strictly confidential) | 1 day |
| normal | confidential | vertraulich | confidential | 1 week |
| low | (restricted) | nur für den Dienstgebrauch | internal | 2 week |
| none | (public) | (öffentlich) | public | N.A. |

## Privacy by Design Aspects

**TRY Processing Transparency**

Full transparency of all person-related data processing actions and the rationales for the processing.

**DIF Differential Privacy**

Introduced randomness into the results of queries on underlying confidential or person-related data to ensure ambiguity in the deduced underlying data.

**CST Processing Consent**

Explicit consent from the data owner for the data processing, including the right to contradict the consent at any later time.

**STR Secure Storage**

Secure storage of all person-related information to prevent data loss or exposure while the information is at rest.

**MIN Data Minimization**

Minimization of the gathered, processed and stored person-related data.

**TRS Secure Transfer**

Secure transfer of all person-related information to prevent data loss or exposure while the information is transferred.

**PSN Data Pseudonymization**

Substitute person-related information with artificial identifiers (pseudonyms) while still keeping information for backtracking to the original information.

**ERA Right to Erasure**

Ensure the data owners, at any time, have the right and operational means to lookup and optionally erase all their person-related information.

**ANO Data Anonymization**

Substitute person-related information with artificial identifiers without keeping any information for backtracking to the original information.

**PRT Right to Portability**

Ensure the data owners, at any time, have the right and operational means to lookup and optionally export all their person-related information.

**AGG Data Aggregation**

Always report person-related statistical information only as aggregated numbers which each does not cover less than about 10 involved persons.

**NOT Breach Notification**

Transparently provide notifications in case of any protection breach of confidential or person-related information.

Intellectual Content: Version 1.0.0 (2018-10-13) Authored 2018 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.0 (2018-10-13) Copyright © 2018 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ARCHITECTURE FUNDAMENTALS

# Security Attacks

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 18.2

## Attack Strategies

**UND Undercover**
Directly attack a particular victim system by infiltrating it through an undercover operation.

**DRT Direct**
Directly attack a particular victim system by leveraging the surprise effect.

**MIM Man-In-The-Middle**
Attack a particular victim system by pretending un unrelated foreign third-party system.

**SCR Scarecrow**
Indirectly attack a particular victim system through a scarecrow system.

**TRD Third-Party**
Attack a particular victim system with an unrelated foreign third-party system.

**SPR Spreading**
Spread out globally and attack as much victim systems as possible.

**PCH Passive Couch**
Passive couch and attack all victim systems coming around by coincidence.

## Attack Approaches

**SOC Social Engineering**
Coaxing passwords and other valuable information from unsuspecting users through seemingly innocent conversation.

**BRF Brute Force**
Cracking passwords and other credentials by trying every word in a dictionary and even every possible combination of characters.

**PHS Phishing**
Acquire information, such as usernames, passwords, and credit card details, by masquerading as a trustworthy entity.

**RAC Race Condition**
Exploit the fact that the processing result of a target is dependent on the sequence or timing of other uncontrollable events.

**SPO Spoofing**
Forge or imitate an object, with the intention to induce someone to accept it as genuine.

**RPL Replay**
Repeat a previously wiretapped valid data transmission to cause a target to (re)accept it as coming from the original sender.

**TRJ Trojan Horse**
Trick someone to install a malware program without noticing, by pretending a harmless program.

**BOT Bots**
Use software programs which pretend to be a human user but execute actions a magnitude faster.

**INJ Code Injection**
Introduce foreign code into a computer program to change its execution by exploiting its problematic processing of invalid input data.

**IAB Interface Abuse**
Intentionally improper and not expected use of a networking or programming interface in order to trick a target into a advantageously state.

**SNF Sniffing**
Passive wiretapping of a foreign communication association.

**HIJ Hijacking**
Active wiretapping and seizing control of a previously established communication association.

**FLD Flooding**
Spread out a large amount of data to targets in order to exceed that data which can be accommodated by the targets.

**TRP Trap**
Establish a harmless looking context which allows to catch a target once it enters the context.

**OVF Overflow**
Send an extraordinarily large amount of input data to a target in order to exceed that data which can be accommodated by the target.

**CRK Cracking**
Modifying code to disable or remove features in order to circumvent copy protections, license checks, etc.

## Attack Goals

**PES Privilege Escalation**
Gain elevated access to and perform unauthorized actions on resources that are normally protected from an application or user.

**IDT Identity Spoofing**
Masquerade as a foreign entity to leverage from their security credentials.

**IGN Information Gain**
Gain access to protected information.

**ICR Information Corruption**
Corrupt or even destroy protected information.

**MEX Malware Execution**
Execute malicious software programs.

**CTL Control Takeover**
Take over the control of a target system.

**DOS Denial of Service**
Services are overwhelmed with large number of dummy requests in order to effectively make the service unavailable for regular requests.

Attack = Strategy x Approach x Goal

Intellectual Content: Version 1.0.2 (2014-08-01) Authored 2012-2014 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.2.1 (2020-06-28) Copyright © 2012-2020 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed for Technische Universität München (TUM) for reproduction in Computer Science lecture courses only.

ARCHITECTURE FUNDAMENTALS

# Security Defenses

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 18.3

## Defenses at Network Protocol Level

### SNP Secure Network Protocol
Use a standardized, proven to be secure, network protocol to protect from Man-in-the-Middle, Undercover, Spoofing, Hijacking, and Replay.
SSL/TLS

### VAL Input Validation
Use smart validation and canonicalization of all inputs to prevent problems during further processing.
JSON Schema

### FTE Faked Temporary Errors
Fake temporary protocol errors to protect from Spreading and flooding Bots.
SMTP Greylisting

### ESC Output Escaping
Use strict escaping of problematic characters in all outputs.
HTML Entity

### TMS Timestamping
Use timestamps or virtual clocks to sequence communication and protect from Replay.
OAUTH

### SIG Digital Signature
Use a cryptographically strong message digest to digitally sign an artifact in order to allow receivers to verify its origin.
RSA+SHA1

### PRX Protocol Proxy
Use application-level, filtering proxies in the communication to protect from Code Injection and Interface Abuse.
PGProxy

### ENC Data Encryption
Use a cryptographically strong cipher to encrypt an artifact in order to allow privacy in communication.
AES

### PID Peer Identification
Use peer identification to distinguish between different peers.
SMTP

### PKI Public-Key Infrastructure
Use private/public keys and a supporting certificate authority infrastructure to leverage from asynchronous cryptography.
X.509

### PAU Peer Authentification
Use cryptographically strong peer identification to protect from Man-In-The-Middle, Spoofing and Hijacking.
SSL/TLS

### OOB Out-of-band Fingerprint
Use message digest based fingerprints to check the integrity of artifacts by transferring the fingerprints via out-of-band communication.
SSH FP

### TFA Two-Factor Authentication
Use two different types of security credentials (which are required at the same time) to authenticate a peer.
RSA Tokens

### WHT Peer Whitelisting
Use a strict and binding list of peer identifiers to restrict communication to the set of listed peers.
ACL

### SID Unspecific Errors
Use less specific and common errors in responses to security-related operations to not indirectly leak sensitive information.
Login

### ZNP Zero Knowledge Protocol
Use an interactive method where one peer proves to another peer that a (usually mathematical) statement is true, without revealing anything other than the veracity of the statement.
SRP

## Defenses at Software Application Level

### EDU User Education
Use explicit security education for end users to prevent Social Engineering based attacks to succeed.
Letters from Bank Institutes

### THR Service Throttling
Use exponential delays to throttle service requests per single communication peer to protect from Denial-of-Service attacks.
Postfix smtpd

### SPP Strong Password Policy
Use a strict password policy to ensure that passwords are as strong as possible to protect from dictionary lookup based Brute Force attacks.
Windows Domain Policies

### FOV Service Failover
Use automatic failover of services to fallback systems to prevent Denial-of-Service attacks.
Load Balancer

### SLT Salted Hashing
Use a random input (salt) to message digests which are stored side-by-side with the resulting hash value in order to prevent dictionary lookup based Brute Force attacks.
Unix crypt

### ATO Atomic Operation
Use atomic operations, which either succeed or fail as a whole, to ensure consistency and protect from Race Condition attacks.
Unix 512b I/O

### TTC Turing Test Completion
Use an Turing Test which a peer has to interactively complete in order to distinguish between a human user and a Bot attack.
CAPTCHA

### SND Execution Sandboxing
Use a constrained and controlled run-time environment to restrict the execution of operations.
JVM

### MAC Mandatory Access Control
Use system-level enforced constraints which restrict the ability to access or generally perform some sort of operation on an object or target.
BSD MAC

### SEP Process Separation
Use separated processes to restrict privileged operations to dedicated constrained processes.
Chrome

### RBA Role-Based Authorization
Use authentication based on distinct user roles to fine-granularly control access to resources and operations.
Linux RBAC

### VER Data Versioning
Use versioning of data to prevent data losses.
MVCC

### BET Bounded Execution Time
Use maximum run-times to bound the execution times of all major operations.
Apache MaxClient

### ENV Sane Environment
Use clean and sane run-time environments to ensure maximum independence from the environment and to ensure deterministic results.
Unix "env -i"

### BRU Bounded Resource Usage
Use maximum run-time resource allocations to bound the overall resource usage.
Unix ulimit

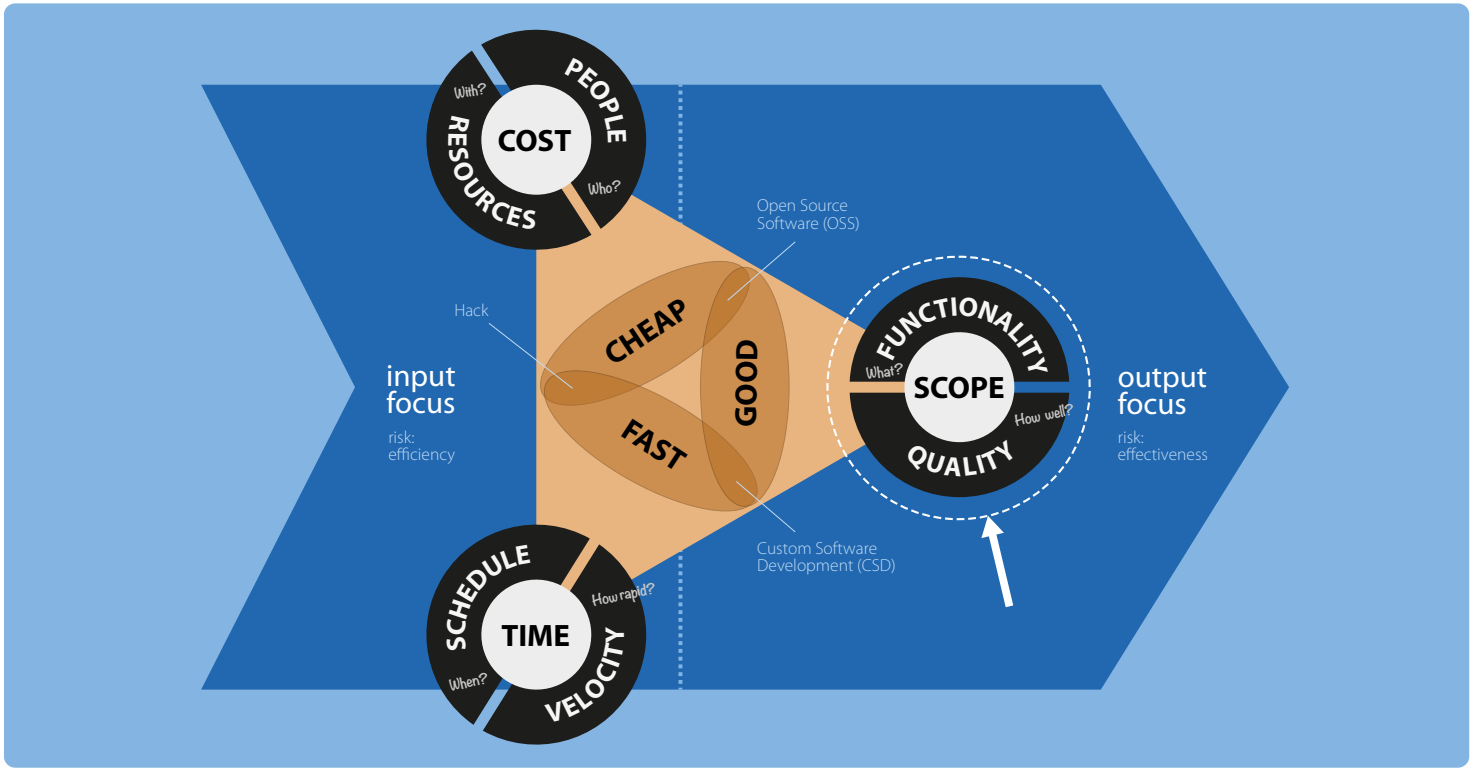
### PST Prepared Statements
Use prepared statements and variable bindings for all backend database communication to protect from Code Injection attacks.
Java PreparedStatement

Intellectual Content Version 1.0.2 (2012-10-17) Authored 2012 by Dr. Ralf S. Engelschall
Graphical Illustration Version 1.0.4 (2020-06-28) Copyright © 2012-2020 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized reproduction/hosting/sale prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

ENGINEERING FUNDAMENTALS

# Project Management Triangle

TECHNISCHE UNIVERSITÄT MÜNCHEN




**Definition of a *Project*:**
"*Temporary endeavor undertaken to create a unique product, service or result.*"
**Temporary** in that it has a defined beginning and end in time, and a defined scope and cost.
**Unique** in that it is not a routine operation, but a one-time, single-goal, and risk-containing operation.

**Project Management *Iron Triangle*:**
A project is constrained by **time**, **cost** and **scope**. No constraint in this triangle can be changed without affecting the others.
Time splits into **schedule** and **velocity**.
Cost splits into **people** and **resources**. Scope splits into **functionality** and result **quality**.

**Project Management *Trilemma*:**
*"Fast. Cheap. Good. Pick two!"*
Each project optimization effort has the choice among **three** favourable options — only **two** of them are possible at the same time.

EF 04.1

*project & constraints*

Intellectual Content: Version 1.1.3 (2021-09-20). Authored 2010-2021 by Dr. Ralf S. Engelschall, based on project management body of knowledge of PMI and IPMA.
Graphical Illustration: Version 1.1.8 (2021-09-11). Copyright © 2010-2021 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contents only.

ENGINEERING FUNDAMENTALS

# Project Management Building Blocks

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 04.2

## STRUCTURE

### Time Periods

**Product Stage** — STG — 1

**Giant** step in the development of the product, to deliver a distinct **major** version of the product.

**Project Phase** — PHS — 2

**Major** time period in the project process, to split the project into distinct **focus periods,** separated by major milestones.

**Project Period** — PRD — 3

**Minor** time period in the project process, to split the project phases into distinct time-, cost-, or scope-based **time units.**

### Work Streams

**Organisation Scope** — SCP — 12

Top-down scopes, Portfolio, Program and Project, to manage initiatives at different granularity levels.

**Product Increment** — INC — 11

**Regular** step in the development of the product, to deliver a distinct **minor** version of the product.

**Disciplines Iteration** — ITR — 9

Work stream consisting of a single sequential or parallel pass over all disciplines.

**Disciplines Track** — TRK — 8

Work stream consisting of a continuous repeated flow through the domain of related disciplines.

**Engineering Discipline** — DIS — 6

Distinct knowledge and work area of Software Engineering.

### Process Flows — 7

**Flow Sequencing** — SEQ

Sequential process flow of multiple disciplines, iterations, tracks, slices, increments, phases or stages.

**Flow Parallelization** — PRL

Parallel process flow of multiple disciplines or tracks.

**Flow Interleaving** — ILV

Interleaved process flow of multiple disciplines, iterations, tracks, slices, increments, phases or stages.

## PROPERTIES

### Progress Modes — 4

**Voluntary** (Supplier-Push) — VOL

**Supplier pushes** requirements and work packages **into** the project and controls the progress.

OSS

**Planned** (Customer-Push) — PLN

**Customer pushes** requirements and work packages **into** the project and controls the progress.

RUP

**Agile** (Supplier-Pull) — AGL

**Supplier pulls** requirements and work packages **out of** the project and controls the progress.

Scrum

**Lean** (Customer-Pull) — LEA

**Customer pulls** requirements and work packages **out of** the project and controls the progress.

Kanban

### Work Focuses — 5

**Project Focus** — PRJ

Focus on the project itself, to initiate, define, plan and successfully close it.

**Technology Focus** — TEC

Focus on the IT technology, to use it for implementing the solution.

**Domain Focus** — DOM

Focus on the domain, to analyse and specify it.

**Environment Focus** — ENV

Focus on the environment of the solution, to transition the solution into it.

### Work Goals — 10

**Extension Goal** — EXT

Following the goal of making a **functional** extension of the product to create a **new** increment.

**Revision Goal** — REV

Following the goal of making a **quality** revision of the product to improve an **existing** increment.

**Reduction Goal** — RED

Following the goal of making a **functional** reduction of the product to destroy an **existing** increment.

**PROCESS BUILDING BLOCKS:**
Every Project Management process in Software Engineering is made of the above building blocks. All building blocks can occur (structure) or be applied (properties) zero, one or more times in a particular process.

**PROCESS TAILORING & CREATION:**
To tailor an existing process, use the defined building blocks to better understand the given process. To create a process from scratch, decide on the building blocks by following steps 1 to 12 in the given order.

Intellectual Content: Version 1.0.8 (2022-08-11) Authored 2016-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.9 (2021-09-11) Copyright © 2016-2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited | **Latest version is always available at the project site at www.tum.de**

*process & building blocks*

ENGINEERING FUNDAMENTALS

# Plan-Driven Project Management

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 04.3

**Goals & Constraints** GNC

Time (Schedule, Velocity), Cost (Human, Resources), Scope (Functionality, Quality)

**Context & Environment** CNE

Domain / Social & Intern / Extern Context / Environment.

**Stakeholders** STH

Name, Role, Interest / Expectation, Interest / Fear, Project Stance, Project Impact, Conflict Potential.

**Risks & Chances** RNC

Risk / Chance, Origin, Impact, Probability, Damage Amount, Risk Level.

**Organization** ORG

Organization Units, People, Roles, Reporting Lines, Steering Lines.

**Phases** PHS

Temporal Phases, Major Milestones.

**Work Breakdown Structure** WBS

WBS Elements, Objects / Functions / Phases, Work Packages.

**Schedule** SCH

WBS Elements, (Time) Effort, Predecessor Elements, Successor Elements, Milestones.

**Resources** RES

WBS Elements, Required Efforts (Person Days Intern / Extern), Required Resources (Costs).

**Financing** FIN

Temporal Phases, Functionalities / Features, Costs, Sources.

**Project Profile** PRO

Customer, Supplier, Project, Project Type, Context, Goal, Budget, Start Time, End Time.

**Steering** STR

Status Determination Approaches, Reports, Communications Paths.

In classic/plan-driven Project Management (PM), the objectives (time, costs, scope) are worked out in detail during project planning, on the basis of the business case and the project goals. Before actual implementation, the necessary processes and management plans are created and recorded.

Only after complete planning and approval of these by the customer does the implementation itself begin. For this purpose, the life-cycle of the project is divided into individual phases with defined milestones. Only after the successful completion of a phase there is a transition to the next phase.

Intellectual Content: Version 1.00 (2022-11-20) Authored 2022 by Dr. Ralf S. Engelschall based on concepts from Steve McDonnell
Graphical Illustration: Version 1.00 (2022-11-20), Copyright © 2022 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

*thinking & planning*

ENGINEERING FUNDAMENTALS

# Agile Project Management

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 04.4

## Agile Manifesto & Scrum Values

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:



| | | |
|---|---|---|
| Individuals and Interactions | over | Processes and Tools |
| Working Software | over | Comprehensive Documentation |
| Customer Collaboration | over | Contract Negotiation |
| Responding to Change | over | Following a Plan |

That is, while there is value in the items on the right, we value the items on the left more."

— *Manifesto for Agile Software Development*
*http://agilemanifesto.org*

Scrum Theory:
Empiricism
Lean Thinking

Scrum Values:
Courage
Focus
Commitment
Respect
Openness

Essence of Agility:
Driving On Sight
Deliver Regularly

**Sprint**

The heartbeat of a Scrum process, where ideas are turned into value. They are fixed length events of 2-4 weeks. A new Sprint starts immediately after the conclusion of the previous Sprint.

## Scrum Roles

### Product Owner

Captures the needs of customers and stakeholders.

Determines delivery date and delivery content.

Is responsible for product and project success on ROI and TCO basis.

Defines and priorizes the Product Backlog, depending on business value.

Accepts or rejects deliverables.

### Developers

Typically a maximum of 8 people.

Is cross-functional, by coping, as a team, all necessary Software Engineering disciplines.

Members are usually assigned full-time, and membership can only change between Sprints.

### Scrum Master

Is responsible for adherence of the process to the Scrum Theory and the Scrum Values, and acts as a coach and mediator.

Helps to remove obstacles, and protects Developers from external disturbances.

Is responsible to drive the learning process and self-management of Developers.

Has no authority to give disciplinary instructions to the Developers.

Should *not* be a Developer in parallel.

## Scrum Events

### Sprint Planning

First meeting (max. 8h) in the Sprint. Developers, with Product Owner, select the highest priority features of Product Backlog. Developers break down features to tasks and estimate efforts. Developers and Product Owner agree on Sprint Goal. Developers also make commitment to Sprint Goal.

### Daily Scrum (Stand-Up)

Short (0,25h) daily meeting during the Sprint, always in same place and at same time. Everyone answers 4 questions: *What have I achieved since the last Daily? What would I like to achieve by the next Daily? What are the obstacles for me to do this? Are there any obstacles to reaching our Sprint Goal?* Discussions only afterwards.

### Sprint Review

Second to last meeting (max. 4h) of the Sprint, before the Spring Retrospective. The Developers present the Increment they have achieved during the Sprint. Stakeholders give feedback. Only real artifacts or "live demonstrations" are allowed. The whole world is invited to participate!

### Sprint Retrospective

Last meeting (max. 3h) in the Sprint. The "Process, Skill and Individual" dimensions are reviewed as well as "Lessons Learned" are discussed. Improvements are addressed immediately. Supports the mantra of Continuous Improvement (KAIZEN).

## Scrum Artifacts & Commitments

### Product Backlog

Emergent, ordered list of what is needed to improve the product, formulated as User Stories and managed by the Product Owner. User Story granularity is from small to large.

### Sprint Backlog

Is managed by the Developers. Contains the User Stories that are planned for implementation in the current Sprint. These are split into tasks by the Developers.

### Increment

Usable and thoroughly verified state of the product toward the Product Goal. Additive to all prior Increments.

### Product Goal

Long-term goal for the product, formulated as a future state of the product, and provided by the Product Owner. Only one Product Goal is worked towards at any time.

### Sprint Goal

Short-term goal for the current Sprint. What is the priority to be achieved in the Sprint? Why should we start the Sprint (Motivation)?

### Definition of Done (DoD)

Formal description of the state of the Increment, when it meets the quality measures required for the product. When a Product Backlog item meets the Definition of Done, an Increment is born.

*flexibility & change*

Intellectual Content: Version 1.00 (2022-11-20) Authored 2022 by Dr. Ralf S. Engelschall based on concepts from Steve McDonnell
Graphical Illustration: Version 1.00 (2022-11-20), Copyright © 2022 Dr. Ralf S. Engelschall <http://engelschall.com>. All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.