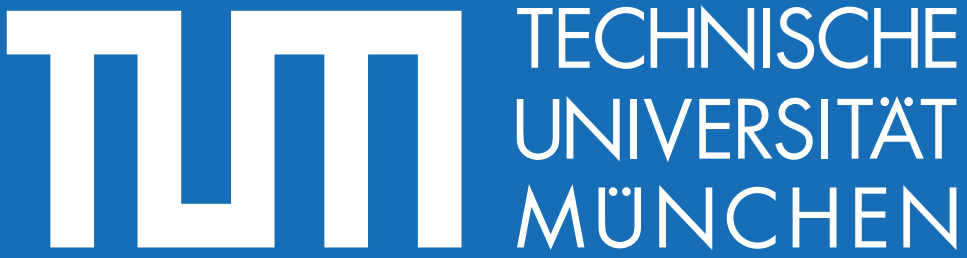
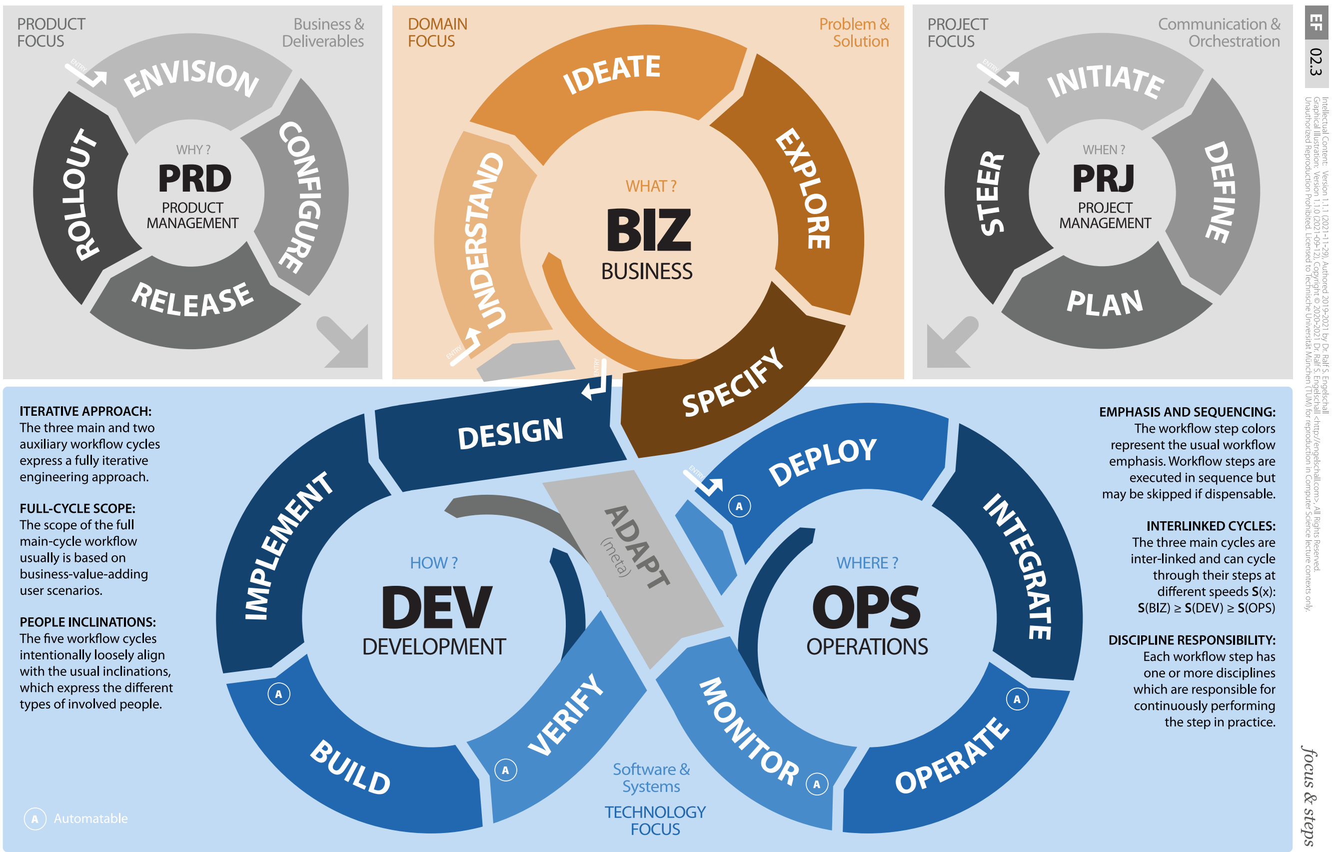


Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall

Software Engineering Workflow



The Workflow Model describes the work segregation in Software Engineering. In the **Workflow**, three main and two auxiliary workflow cycles express the iterative approach. The full main-cycle workflow is usually based on business-value-adding user scenarios.

The three main cycles are interlinked and can cycle through their steps at different speeds, S(x), with S(BIZ) is greater or equal S(DEV), which in turn is greater or equal S(OPS). Because the cycles with earlier steps should not slow down the cycles not later steps.

The step colors represent workflow emphasis. Workflow steps are executed in sequence but may be skipped if dispensable.

The ten discipline areas of Software Engineering express the different roles of involved people. The five inclinations express the different types of involved people. Hence the Workflow consists of exactly five cycles.

Questions

- What does the Workflow-Model of Software Engineering describe?

ENGINEERING FUNDAMENTALS

Software Engineering Steps

TECHNISCHE UNIVERSITÄT MÜNCHEN

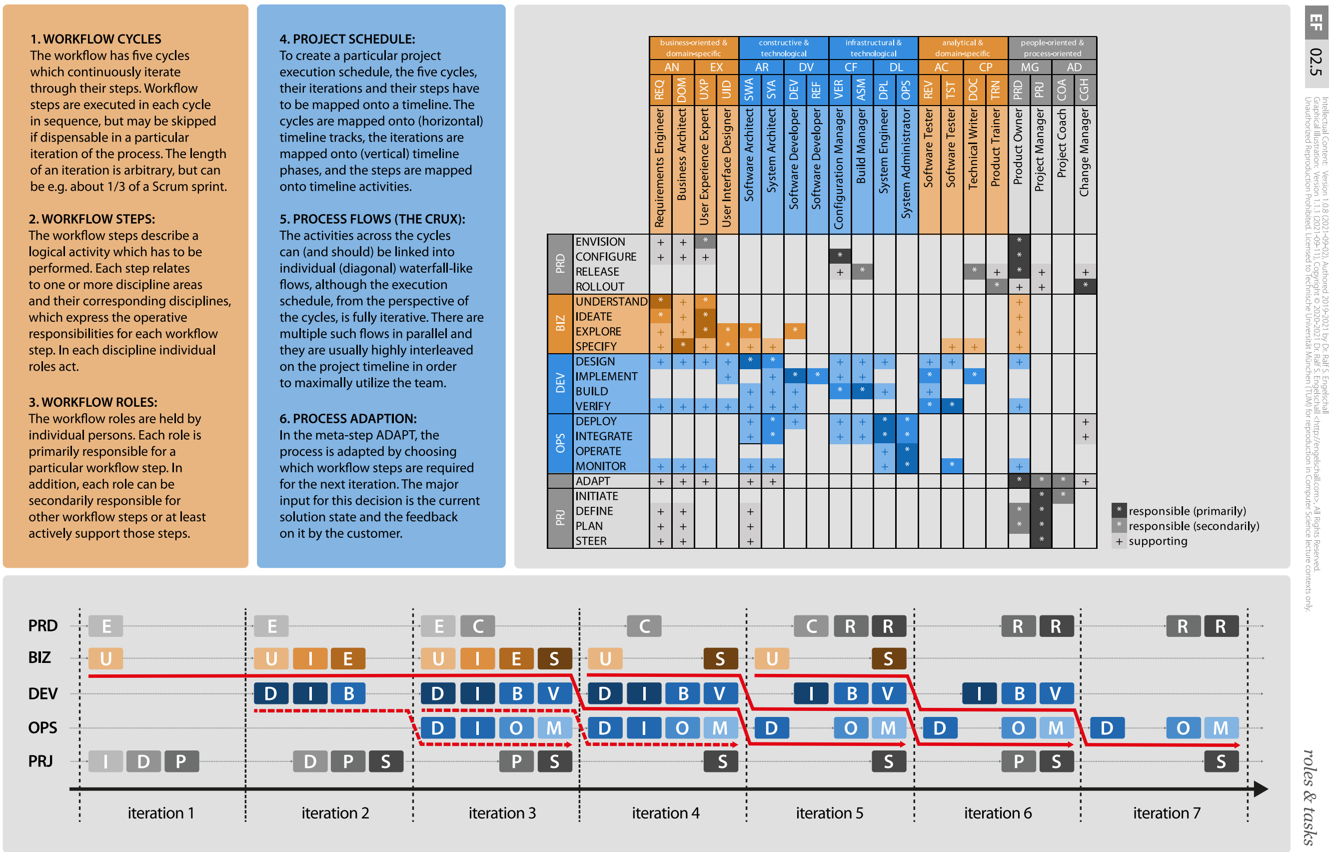
Software Engineering, on an operational level, can be alternatively understood through 20 distinct **Steps** which are continuously performed within the **Software Engineering Workflow**. Each Step belongs to one primarily responsible Discipline and zero or more secondarily responsible Disciplines.

Workflow Steps are the adequate concept to understand which activities have to be performed in each iteration of a **Software Engineering Process**.

Questions

- Which concept allows one to best understand which activities have be performed in a Software Engineering Process?

Software Engineering Process



The workflow has five main cycles which continuously iterate through their steps. Workflow steps are executed in each cycle in sequence but may be skipped if dispensable.

The workflow steps are annotated with discipline areas to express operative responsibilities. In each area, multiple roles act.

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.

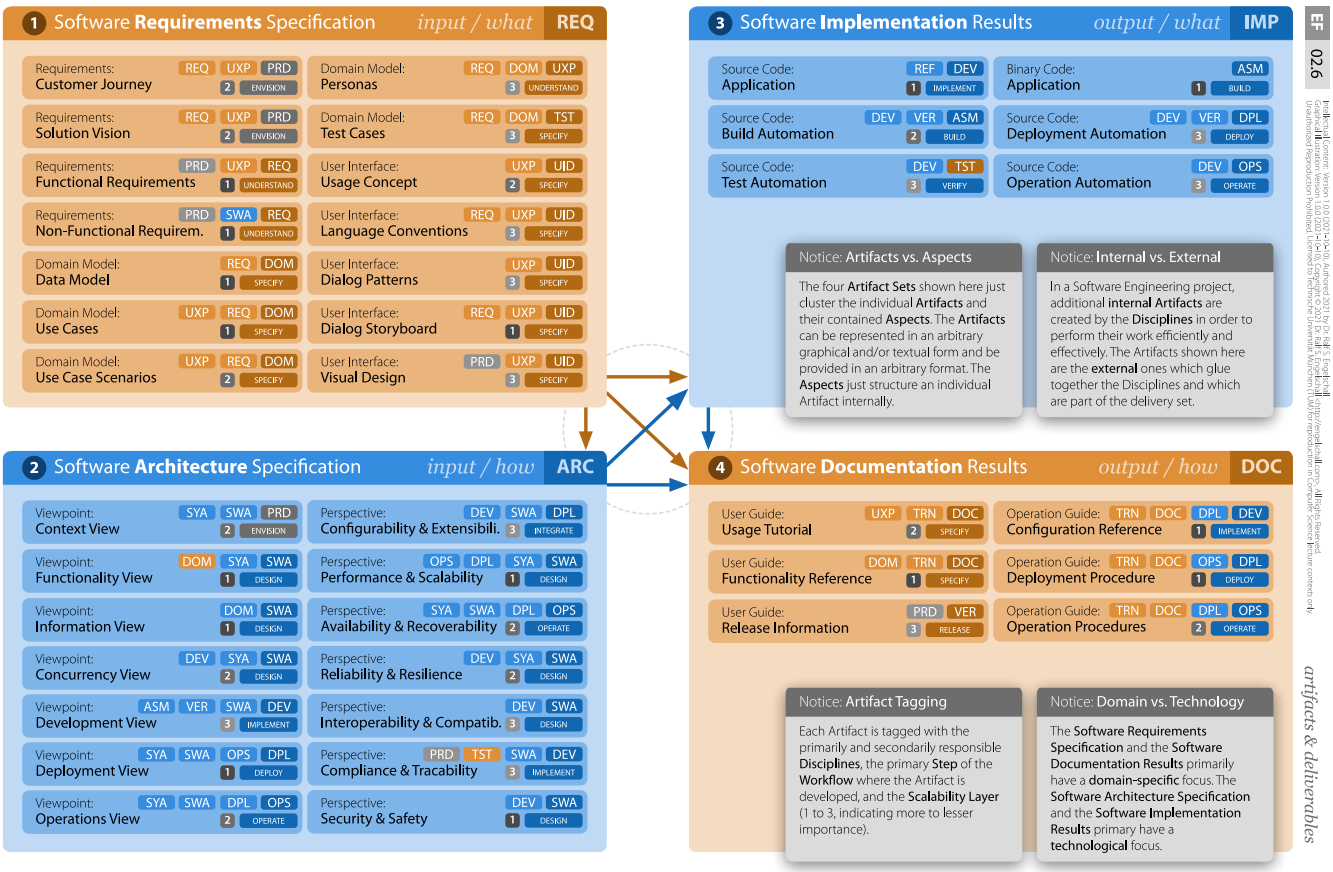
To create a particular project execution schedule, the five cycles, their iterations and their steps have to be mapped onto a timeline. The cycles are mapped onto (horizontal) timeline tracks, the iterations are mapped onto (vertical) timeline phases, and the steps are mapped onto timeline activities.

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.

Questions

- How can maximum utilization of the team be achieved in Software Engineering, despite a division of labor?

Software Engineering Artifacts



The four **Artifact Sets** 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.

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

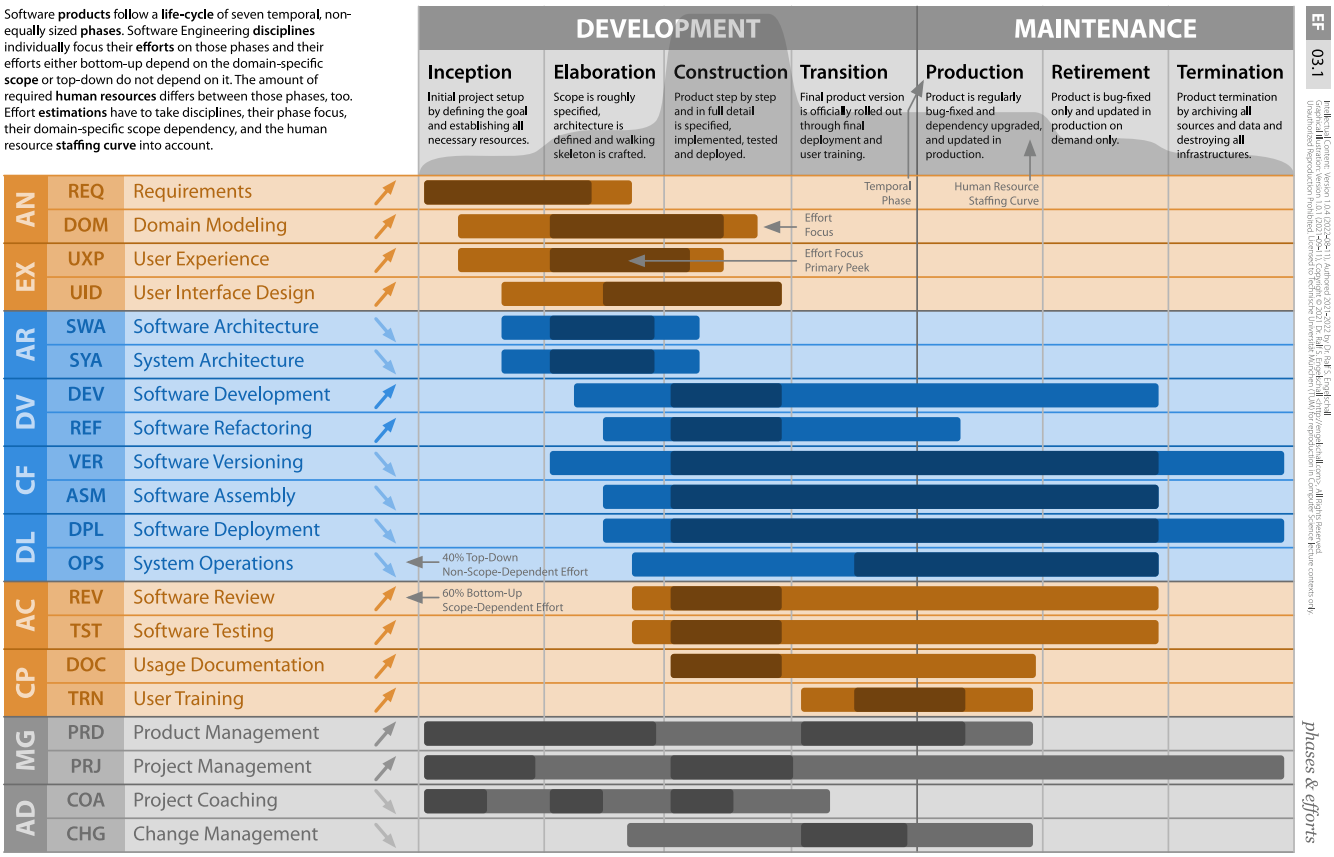
Each **Artifact** is tagged with the primarily and secondarily responsible **Disciplines**, the primary **Step** of the Workflow where the Artifact is developed, and the Scalability Layer (1 to 3, indicating more to lesser importance).

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.

Questions

- What focus has the **Software Requirements Specification**?

Software Engineering Efforts



phases & efforts

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 their efforts 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.

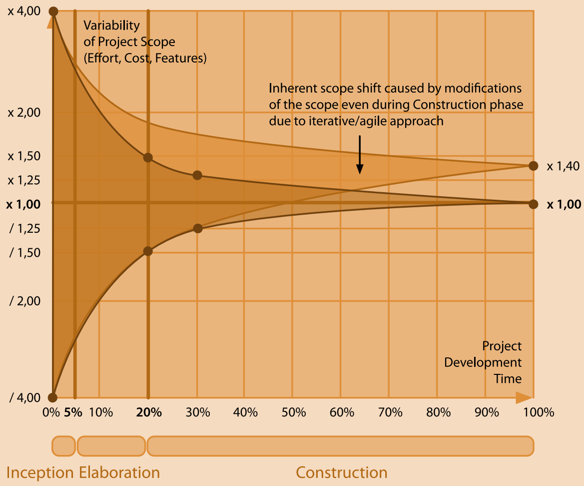
Furthermore, the seven sequential phases especially do not conflict with agile process models: agile time **periods** (named "sprints" in Scrum) merely subdivide the individual phases.

Questions

- What is the Software Engineering **Phase** called, which has the greatest personnel requirements and in which primarily the functionalities are realized?

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.

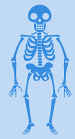
The early development phases Inception and Elaboration especially have to ensure that within the first 20% of the project, the variability is reduced noticeably to just +/- 50%. During the initial iterations of the Construction phase within the first 30% of the project, the variability usually can be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

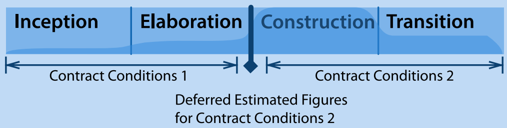
Essential Elaboration Phase

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.

The **Cone of Uncertainty** tells how the variability of the project scope (measured in Effort, Cost or Features) in Software Development changes over time. The early development phases Inception and Elaboration especially have to ensure that the variability is reduced noticeably.

For iterative/agile approaches, experience showed that during the Construction phase, inherently the final scope further shifts due to the just step-by-step learned required details of the required solution.

The Elaboration phase is especially important for the creation of the **Walking Skeleton**, where all the technical integrations of libraries, frameworks, build procedures, etc., are done without already implementing any domain-specific functionalities.

Because of the **Cone of Uncertainty**, **Agile Fixed-Price** project contracts usually differentiate between the early phases Inception and Elaboration and the main phases Construction and Transition. The contract conditions of the latter usually depend on figures which seriously can only be estimated at the end of the Elaboration phase.

Questions

- What is especially developed in the project phase "Elaboration"?

ARCHITECTURE FUNDAMENTALS

Requirements Basics

TECHNISCHE UNIVERSITÄT MÜNCHEN

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>

The **Requirements Specification** is a binding document in which primarily the WHAT and WHY of the solution is specified, however not the concrete technical HOW. The set of requirements must be correct, unambiguous, complete, consistent, prioritized, verifiable, changeable and traceable.

There are two types of requirements: **Functional Requirement** ("Shall Do", functionality) and **Non-Functional Requirements** ("Shall Be", Conditions, in English often expressed with words ending in "-ility"). The architect primarily takes care of the latter.

Requirements can also be reciprocally positive (backing) or negative (trade-off). The architect also primarily takes care of the latter.

Requirements should be "SMART": **Specific**, **Measurable**, **Achieveable**, **Relevant** and **Time-Bound**.

In addition, requirements are either **Enduring** (fixed) or **Volatile** (unstable). The architect should pay attention to the latter.

Questions

? What kind of **requirements** should the architect primarily keep in mind and should be explicitly addressed by him in the solution finding?

Non-Functional Requirements

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 Reproducibility** — 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

…

There are potentially many **Non-Functional Requirements.** For any solution, one must therefore first determine the actual quantity of such Requirements. This quantity must be minimized by the Architect!

For every contractually stipulated Requirement, one should take into account that it is clearly defined since there are great similarities between Requirements and the differences are sometimes very subtle.

A few of the Non-Functional Requirements that almost always have to be considered in practice are **Maintainability**, **Usability**, **Security**, **Availability**, **Reliability**, **Performance**, **Responsiveness** and **Adaptability**.

Questions

- Name 3 in practice frequently considered **Non-Functional Requirements!**