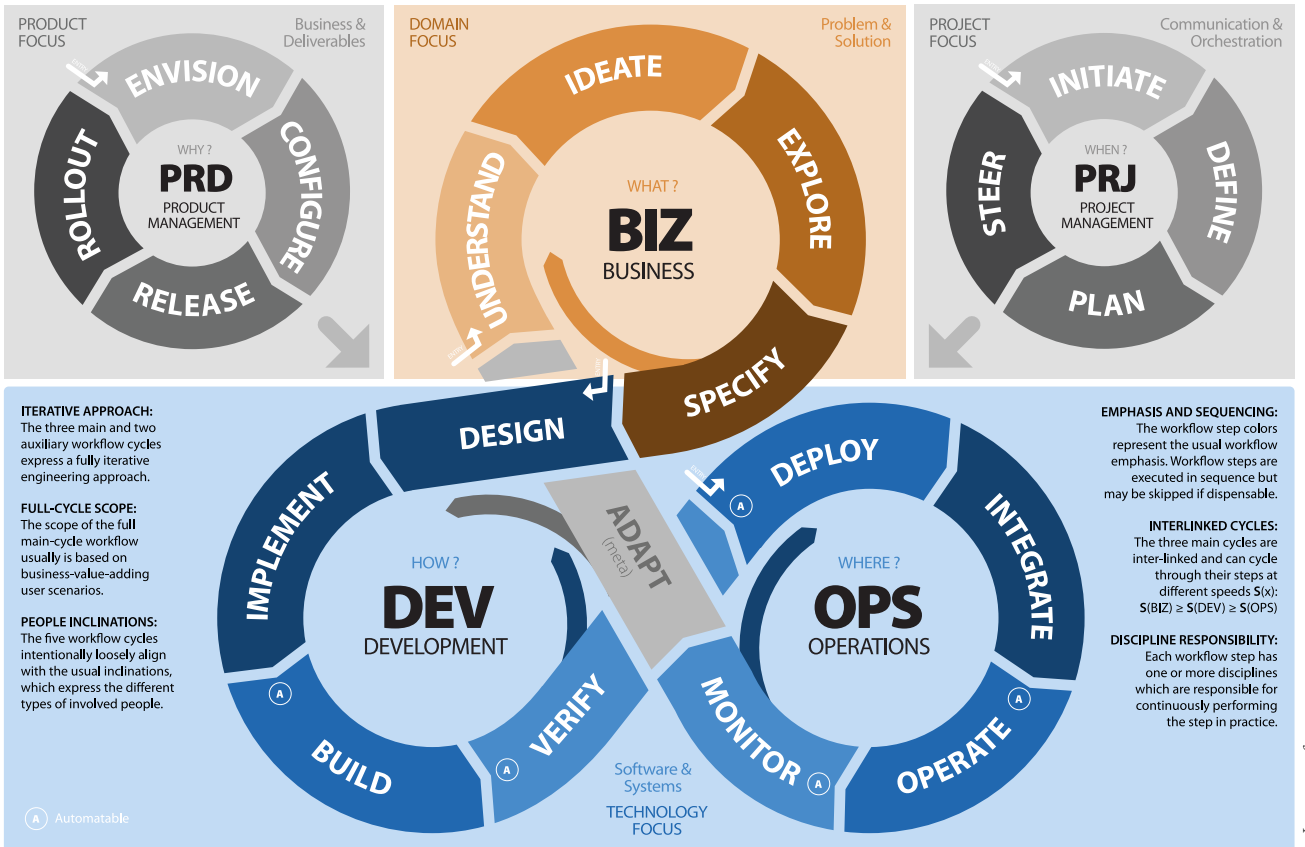




**Software Engineering  
in der industriellen Praxis  
(SEIP)**

Dr. Ralf S. Engelschall



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?



**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.

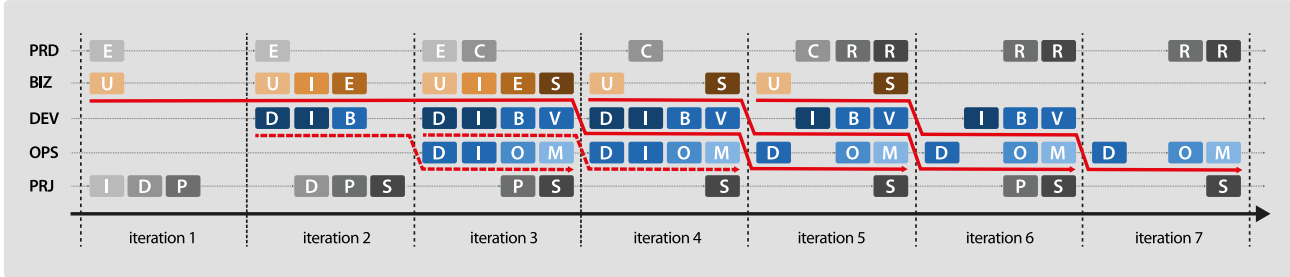
**4. PROJECT SCHEDULE:**  
To create a particular project execution schedule, the five cycles, their iterations and their steps have to be mapped onto a timeline. The cycles are mapped onto (horizontal) timeline tracks, the iterations are mapped onto (vertical) timeline phases, and the steps are mapped 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			professionalized & process-oriented				
	AN	EX	UXP	UID	SWA	AR	DV	CF	DL	ASW	REV	AC	CP	TRM	PRD	MG	AD	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																				
PRJ																				
ADAPT	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
INITIATE																				
DEFINE	+	+	+	+	+															
PLAN	+	+	+	+	+															
STEER	+	+	+	+	+															

responsible (primarily)  
 responsible (secondarily)  
 supporting



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?

**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 Requir.	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

**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.

**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 1 IMPLEMENT
Viewpoint: Operations View	SYA SWA DPL OPS 2 OPERATE	Perspective: Security & Safety	DEV SWA 1 DESIGN

**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** primarily have a **technological** focus.

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.

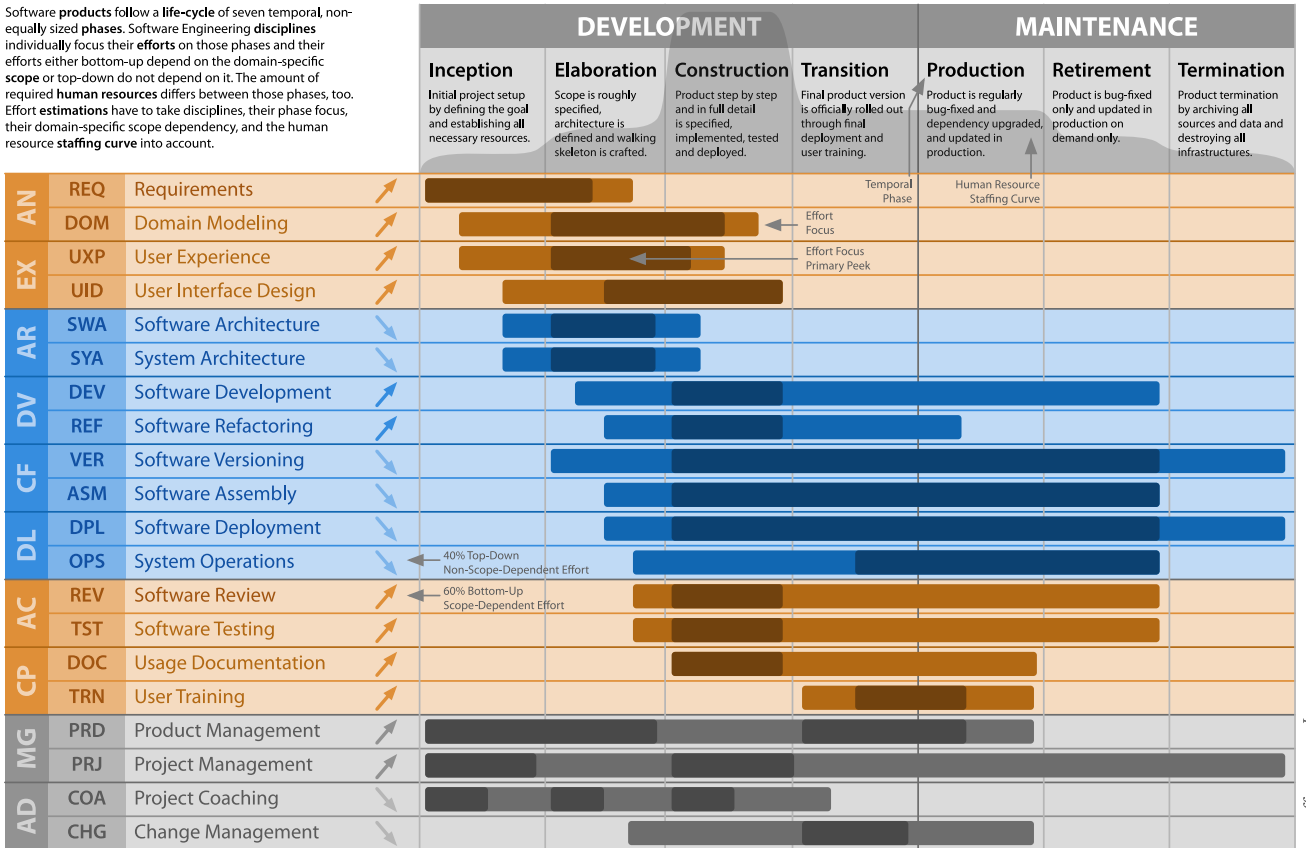
Each **Artifact** is tagged with the primarily and secondarily responsible **Disciplines**, the primary **Step** of the Workflow where the Artifact is developed, and the Scalability Layer (1 to 3, indicating more to lesser importance).

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 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.



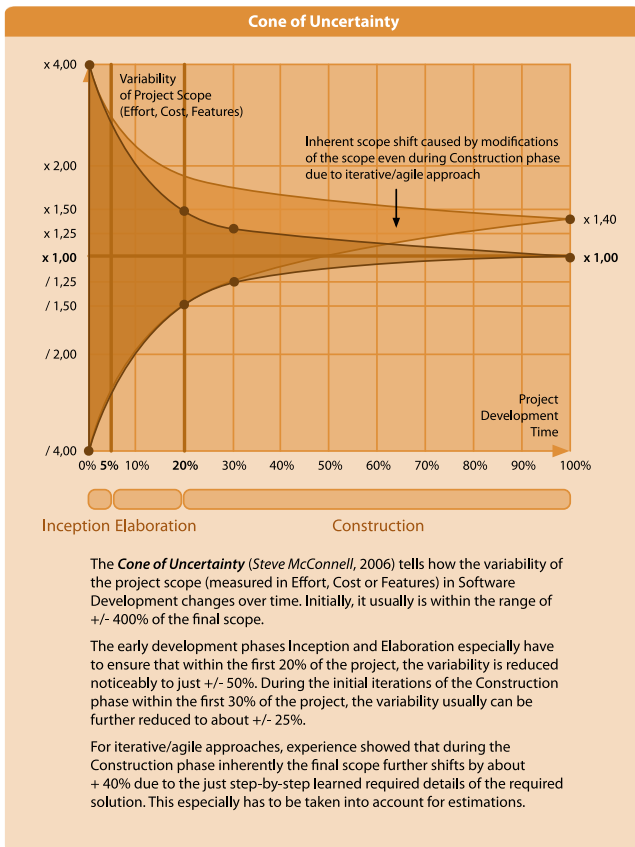
Software products follow a **life-cycle** of seven temporal, non-equally sized **phases**. Software Engineering **disciplines** individually focus their **efforts** on those phases, and their efforts either bottom-up depend on the domain-specific scope, or their efforts top-down do not depend on it. 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?



**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"?



### 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)	NFR Non-Functional (Shall Be)
<p>A condition or capability that a solution must have to provide its service in terms of its behaviour and information. Think: Functionality.</p> 	<p>A condition, property or quality that a solution must have to satisfy a contract, standard, or other formally imposed obligation. Think: Constraints and “-ilities”.</p> 



### Requirement Interdependencies

POS Positive (Backing)	NEG Negative (Trade-Off)
<p>One requirement supports the other (e.g. for NFRs: Maintainability and Comprehensibility usually support Adaptability, Portability, Modifiability, etc., and Scalability usually supports Availability, etc.)</p> 	<p>One requirement interferes with the other (e.g. for NFRs: Security usually interferes with Efficiency, Usability, Performance, etc., and Orthogonality can interfere with Usability)</p> 

### 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.

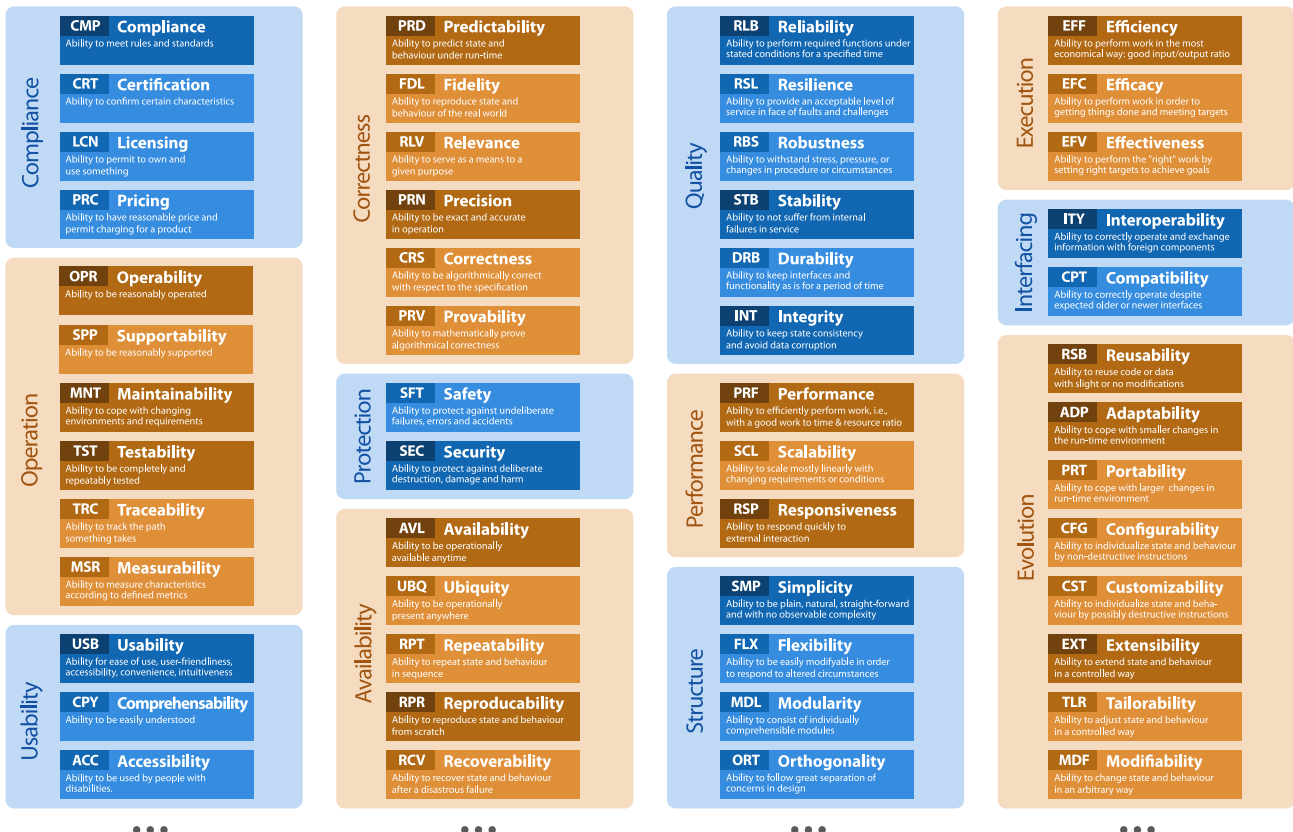
Requirements should be “SMART”: **Specific, Measurable, Achievable, Relevant and Time-Bound.**

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?





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!**