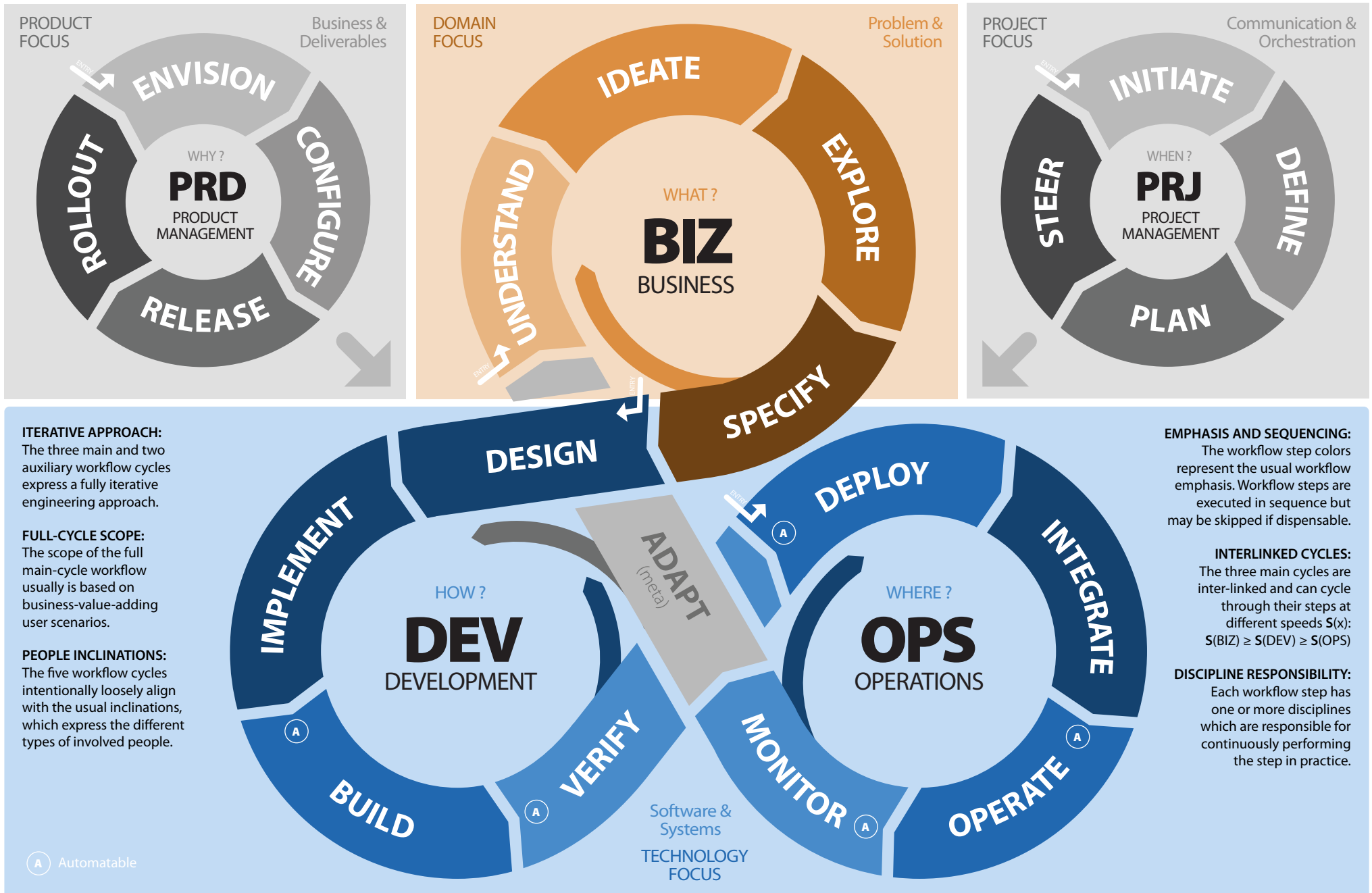




# Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall



# Software Engineering Steps

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.

## 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)

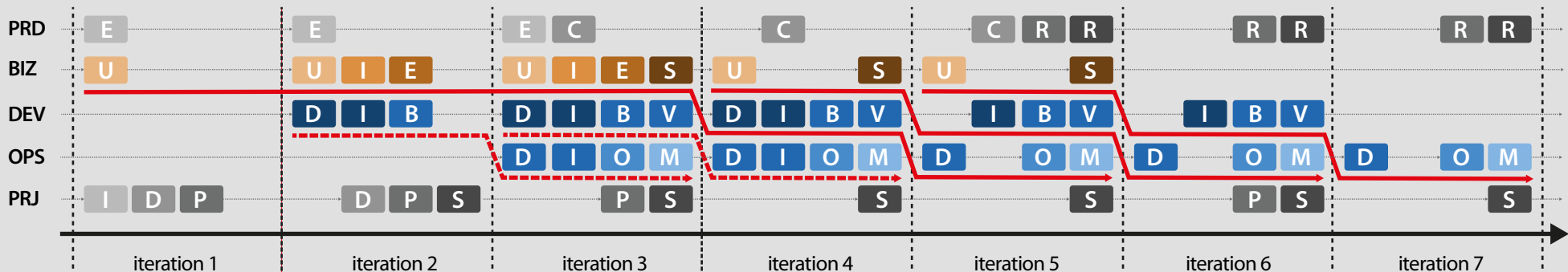
\*

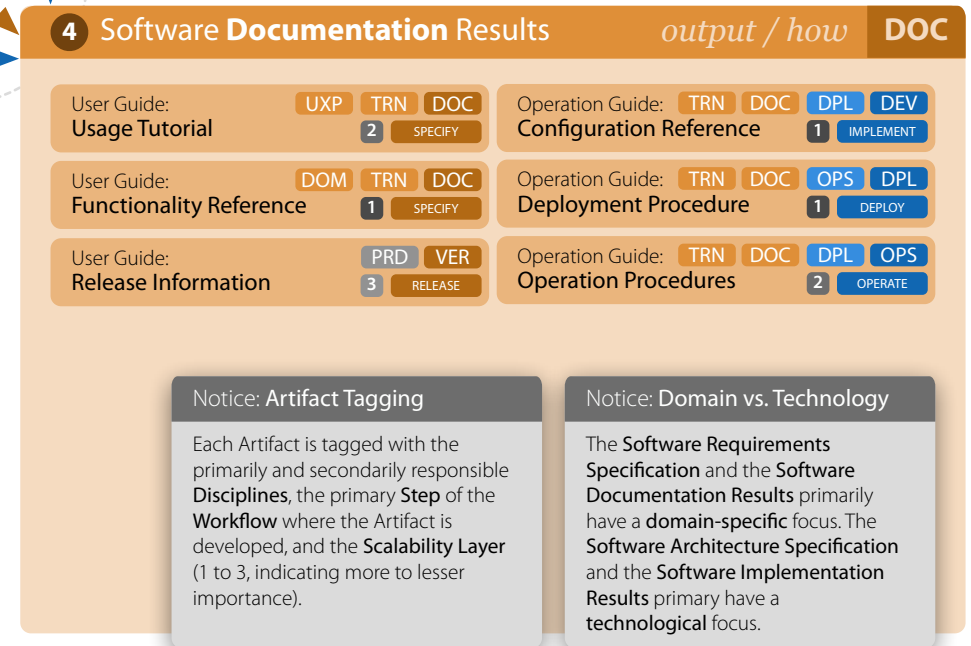
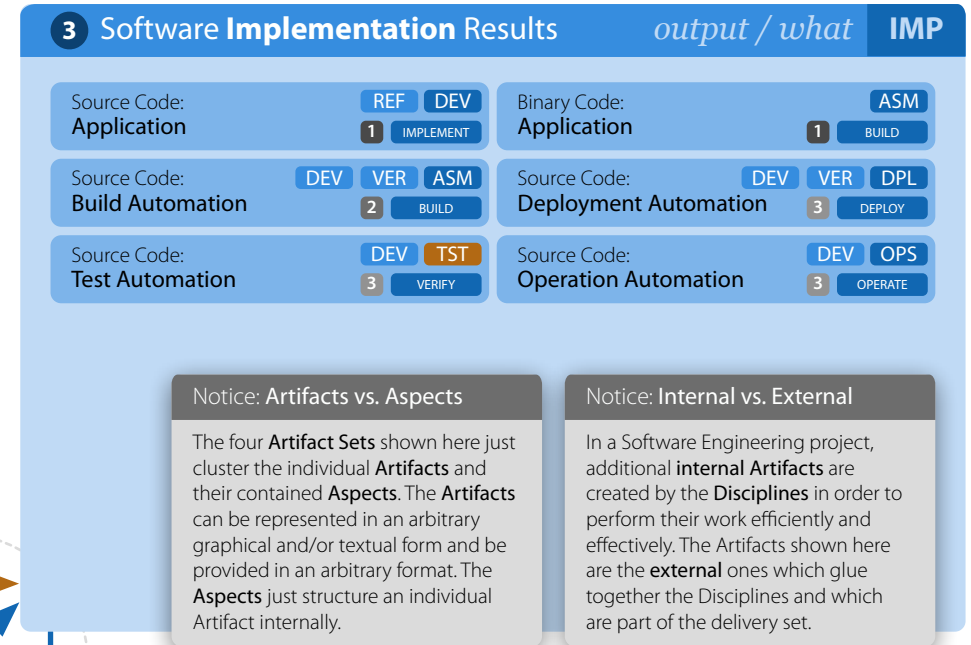
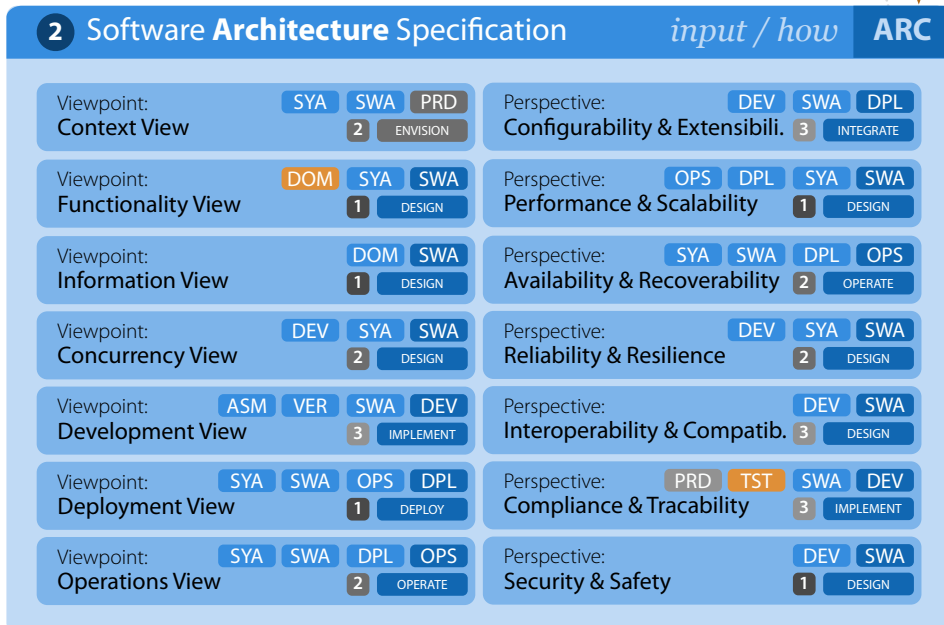
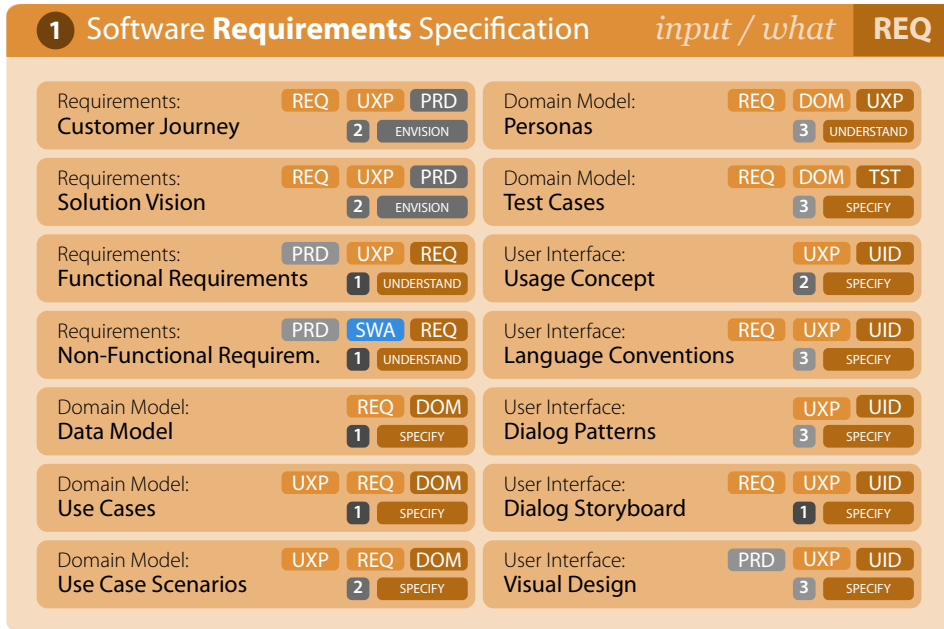
 responsible (secondarily)

+

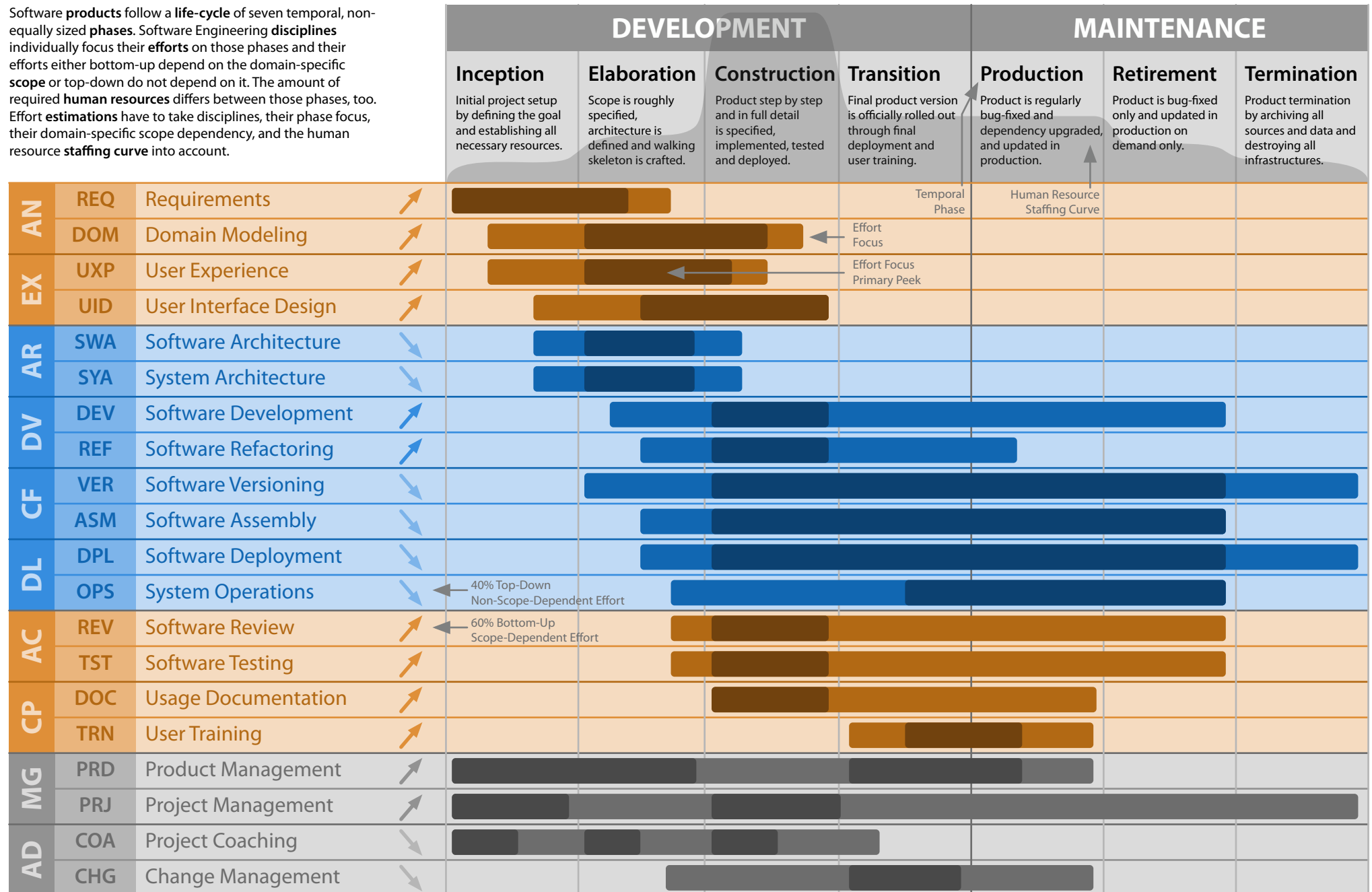
 supporting

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

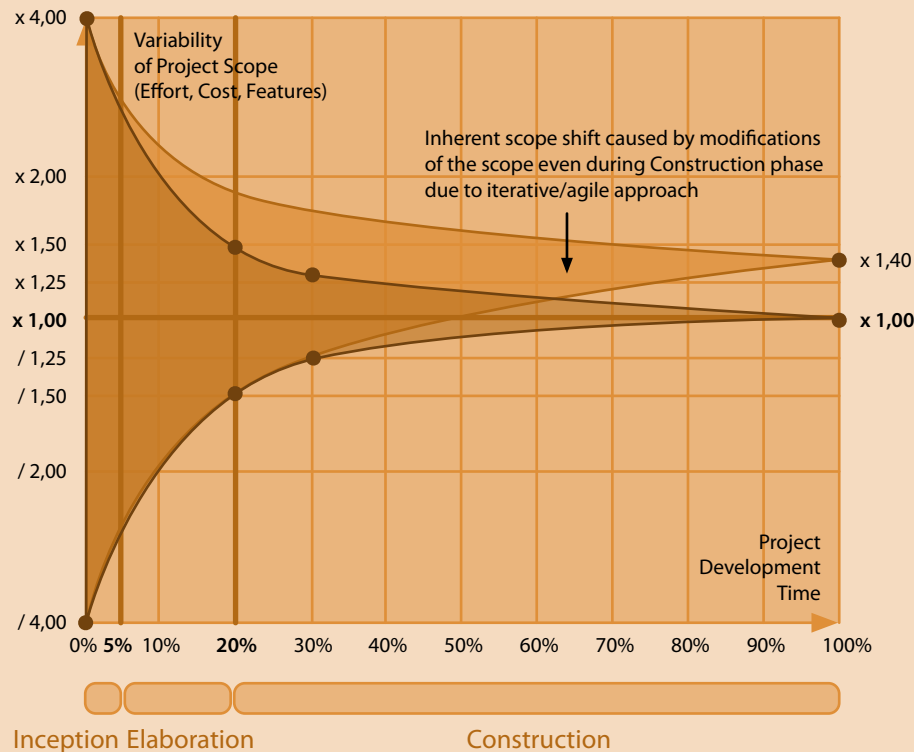




Software **products** follow a **life-cycle** of seven temporal, non-equally sized **phases**. Software Engineering **disciplines** individually focus their **efforts** on those phases and their efforts either bottom-up depend on the domain-specific **scope** or top-down do not depend on it. The amount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



## 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.

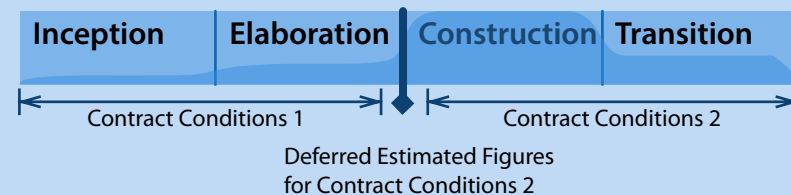
## Essential Elaboration Phase

### 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.



## 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>





