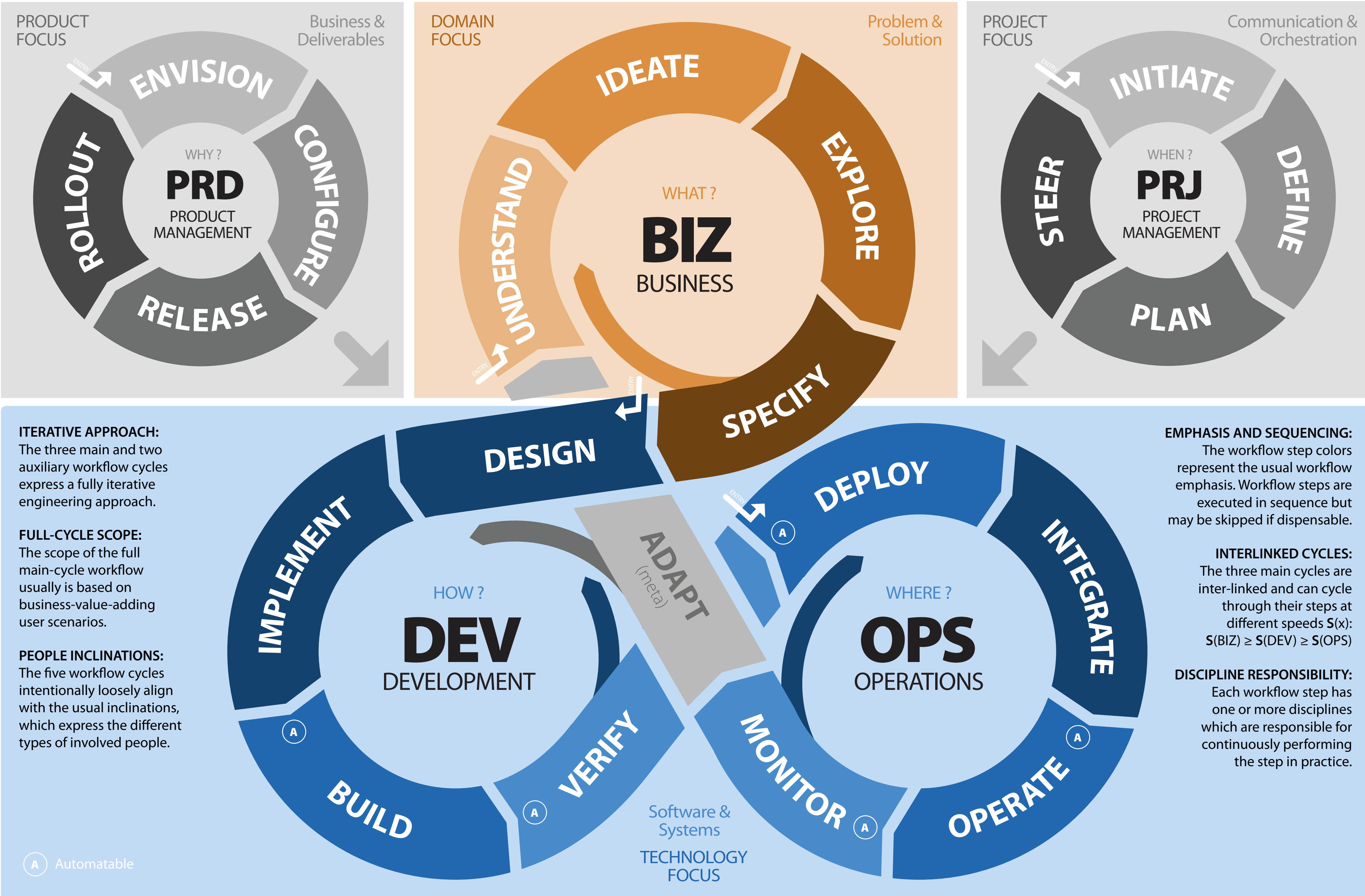
TECHNISCHE UNIVERSITÄT MÜNCHEN

Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall

Software Engineering Workflow



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.

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.

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

ENGINEERING FUNDAMENTALS

Software Engineering Process

TECHNISCHE UNIVERSITÄT MÜNCHEN

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.

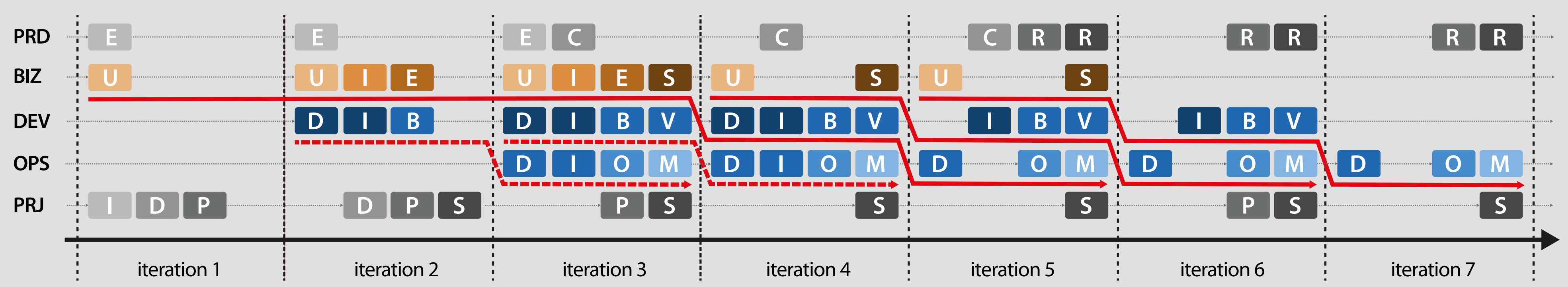
4. PROJECT SCHEDULE:
To create a particular project execution schedule, the five cycles, their iterations and their steps have to be mapped onto a timeline. The cycles are mapped onto (horizontal) timeline tracks, the iterations are mapped onto (vertical) timeline phases, and the steps are mapped onto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

EF 02.5

ENGINEERING FUNDAMENTALS

Software Engineering Artifacts

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 02.6

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

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

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

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

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

Artifact	Disciplines	Layer	Step
Source Code: Application	REF DEV	1	IMPLEMENT
Source Code: Build Automation	DEV VER ASM	2	BUILD
Source Code: Test Automation	DEV TST	3	VERIFY
Binary Code: Application	ASM	1	BUILD
Source Code: Deployment Automation	DEV VER DPL	3	DEPLOY
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

Artifact	Disciplines	Layer	Step
User Guide: Usage Tutorial	UXP TRN DOC	2	SPECIFY
User Guide: Functionality Reference	DOM TRN DOC	1	SPECIFY
User Guide: Release Information	PRD VER	3	RELEASE
Operation Guide: Configuration Reference	TRN DOC DPL DEV	1	IMPLEMENT
Operation Guide: Deployment Procedure	TRN DOC OPS DPL	1	DEPLOY
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.

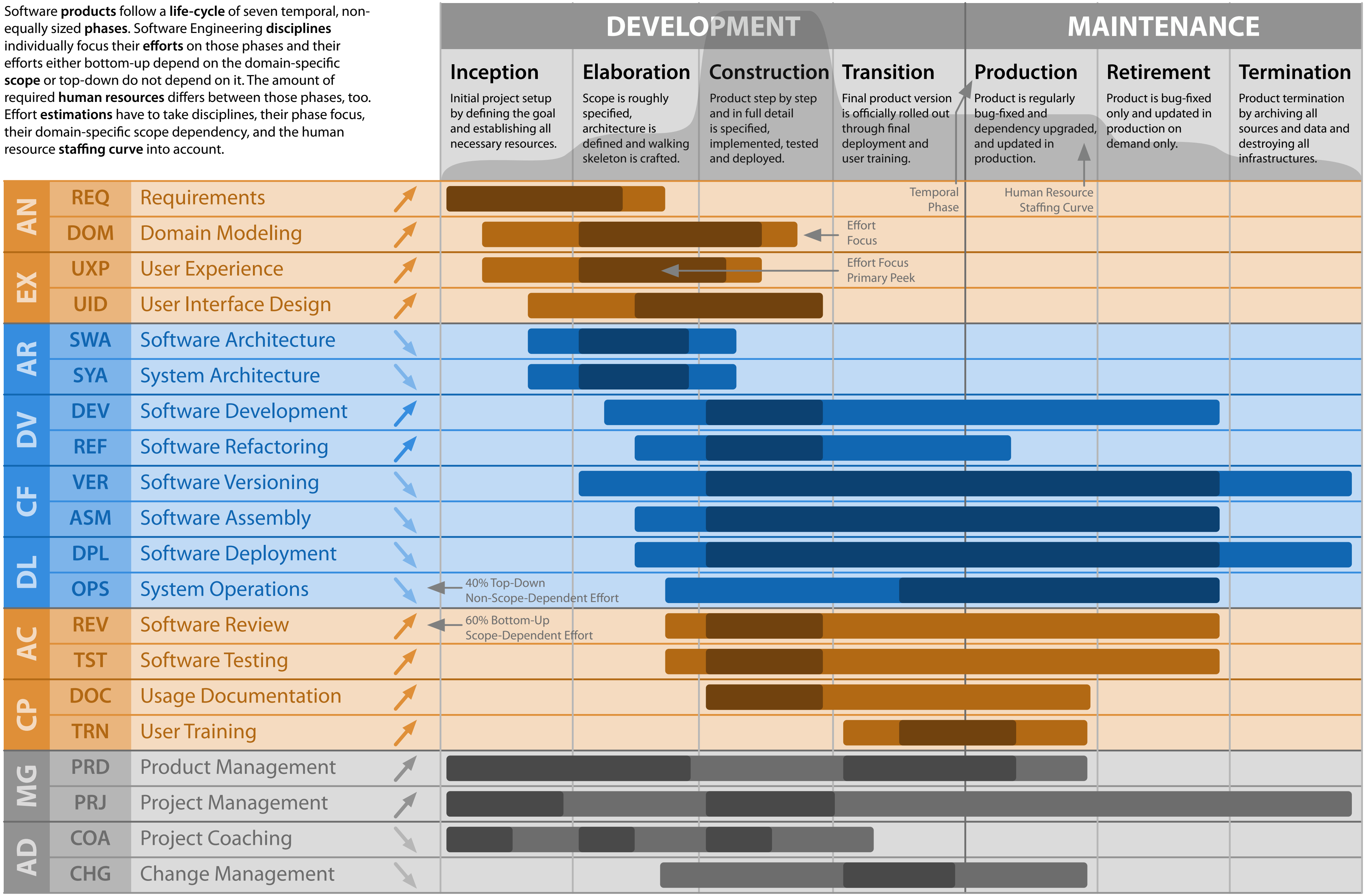
artifacts & deliverables

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

phases & efforts

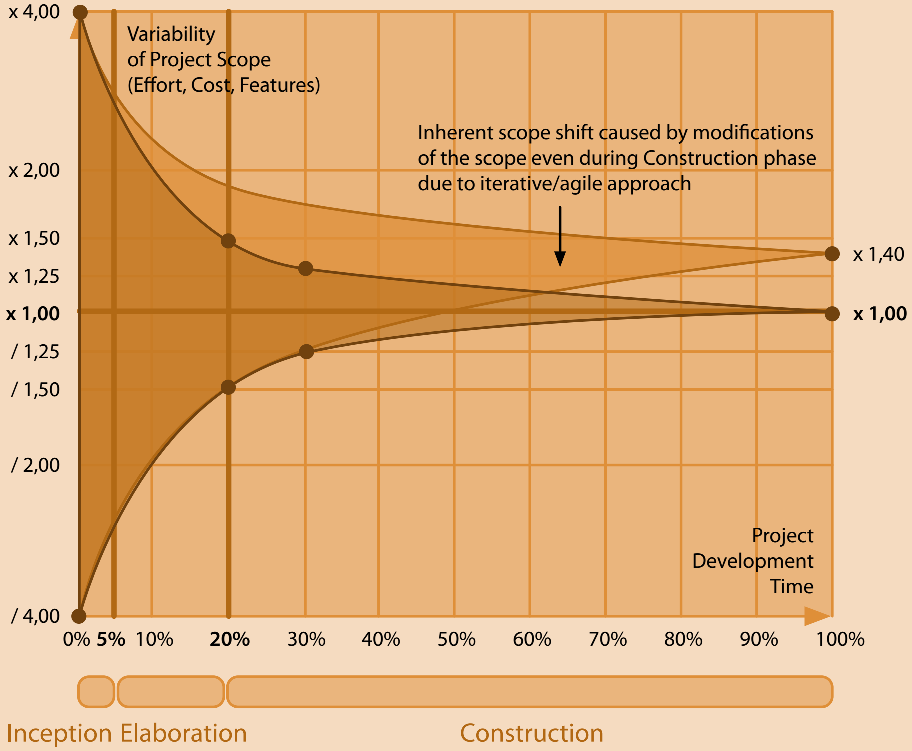
ENGINEERING FUNDAMENTALS

Uncertainty & Elaboration

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 03.2

Cone of Uncertainty



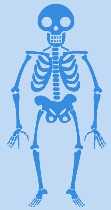
The ***Cone of Uncertainty*** (*Steve McConnell*, 2006) tells how the variability of the project scope (measured in Effort, Cost or Features) in Software Development changes over time. Initially, it usually is within the range of +/- 400% of the final scope.

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.

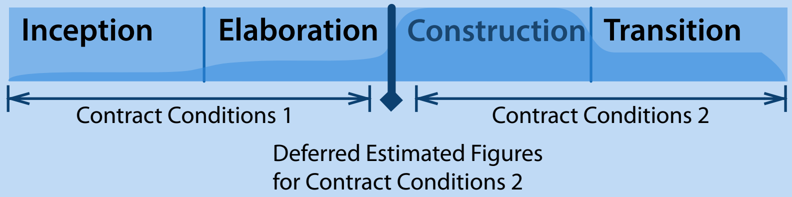
Essential Elaboration Phase

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.

uncertainty & 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>

ARCHITECTURE FUNDAMENTALS

Non-Functional Requirements

TECHNISCHE UNIVERSITÄT MÜNCHEN

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

…