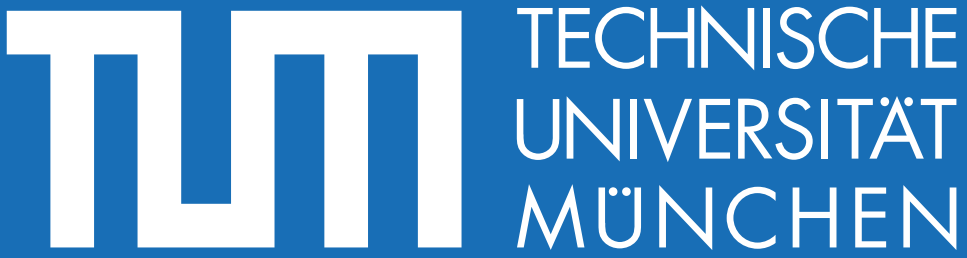
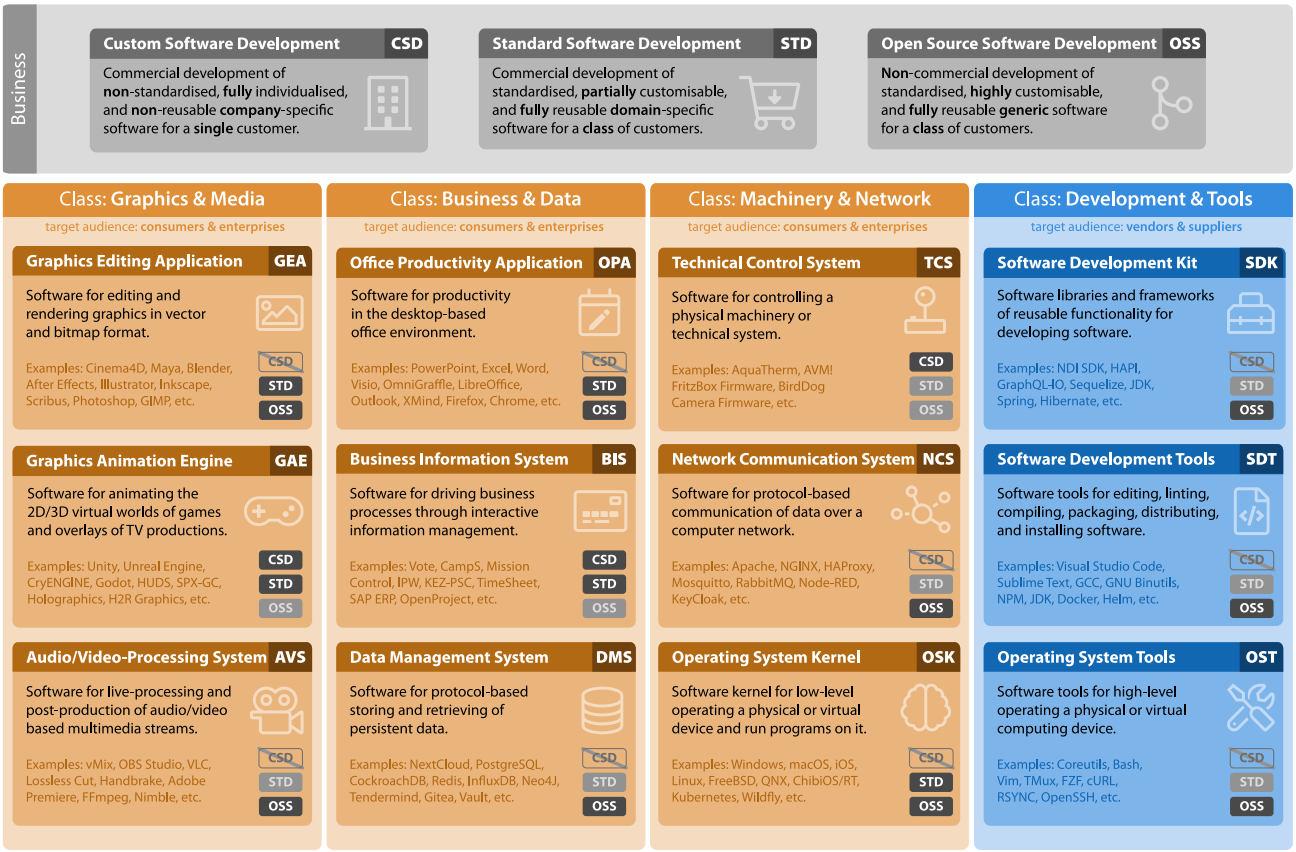


Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall

Software Classes



There are three traditional approaches to Software Development: **Custom Software Development**, the commercial development of nonstandard, fully individualized, and non-reusable company-specific software for a single customer; **Standard Software Development**, the commercial development of a standardized, partially customizable, but fully reusable domain-specific software for a class of customers; and **Open-Source Software Development**, the non-commercial development of a standardized, highly customizable, and fully reusable technical software for any class of customers.

There are three classes of software focusing on Graphics & Media: **Graphics Editing Application**, Software for editing and rendering graphics in vector and bitmap format; **Graphics Animation Engine**, software for animating the 2D/3D virtual worlds of games and overlays of TV productions; and **Audio/Video-Processing Systems**, software for live-processing and post-production of audio/video based multimedia streams.

There are three classes of software focusing on Business & Data: **Office Productivity Application**, software for productivity in the desktop-based office environment; **Business Information System**, software for driving business processes through information management; and **Data Management System**, software for storing and retrieving persistent data.

There are three classes of software focusing on Machinery & Network: **Technical Control System**, software for controlling a physical machinery or technical system; **Network Communication System**, software for communicating data over a computer network; and **Operating System Kernel**, software kernel for low-level operating a physical or virtual computing device.

There are three classes of software focusing on Development & Tools: **Software Development Kit**, software libraries and frameworks of reusable functionality for developing software; **Software Development Tools**, software tools for editing, linting, compiling, packaging and installing software; and **Operating System Tools**, software tools for high-level operating a physical or virtual computing device.

Questions

- Which two classes of software are primarily developed using the **Custom Software Development** approach?

Software Development Approaches

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

goal & approach

One can distinguish four kinds of Software Development approaches.

In **Software Prototyping**, one develops an early sample or model of a software solution by mocking and cheating in order to just once test a concept, idea or process.

In **Software Bricolage**, one develops a single instance of a software solution by tinkering, cobbling, and integrating partial solutions in order to prove feasibility or just provide a service.

In **Software Craftsmanship**, one develops a production-grade software solution by professional, clean but plain craftsmanship means in order to solve a usually complicated problem.

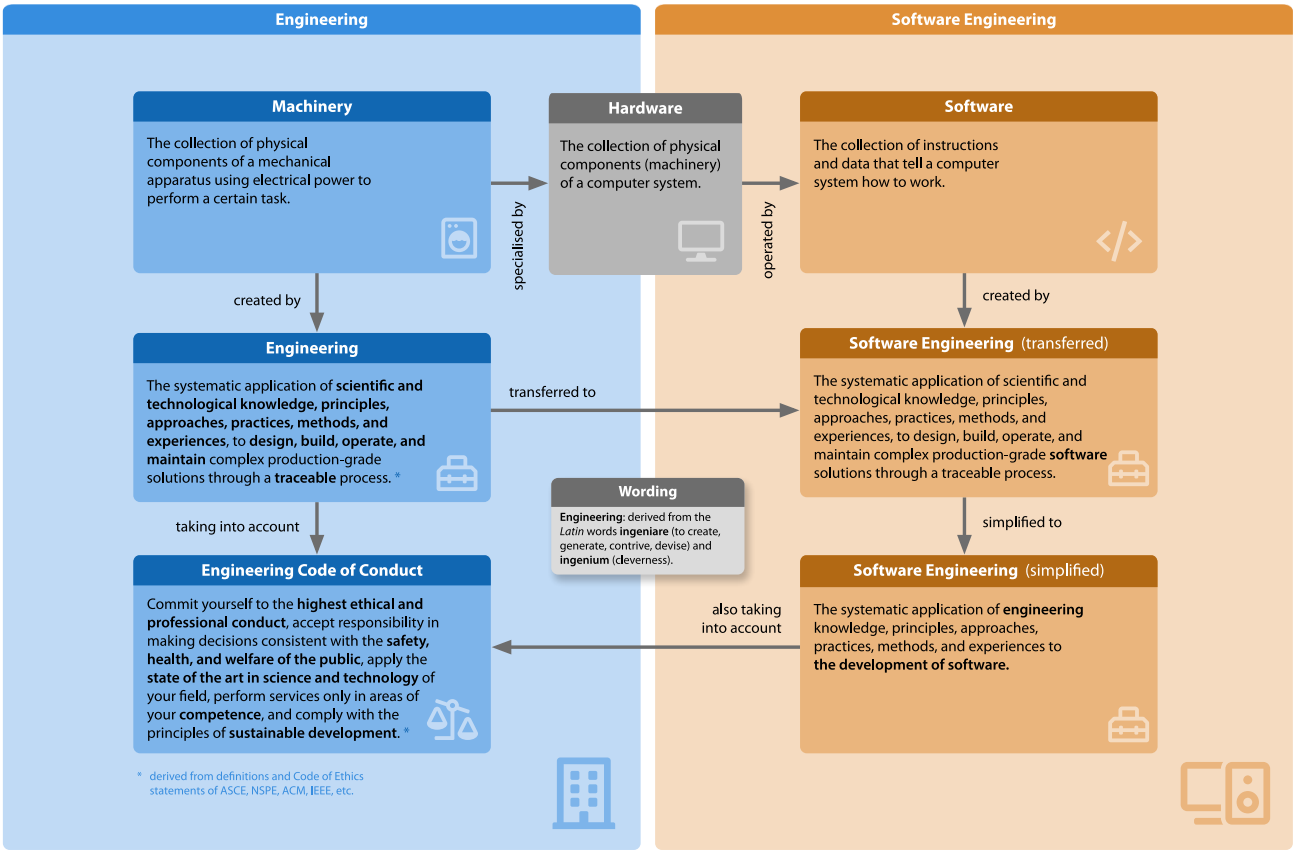
In **Software Engineering**, one develops a production-grade software solution by a professional, risk-hedged engineering approach in order to solve a usually complex problem.

The four development approaches can be combined in practice: Prototyping and Bricolage can be earlier stages of Craftsmanship or Engineering. Craftsmanship can be part of Engineering. Each approach requires a special skill. All four approaches are equally essential in practice. Which one(s) to choose entirely depends on the particular requirements.

Questions

- Which Software Development Approach should be choosen to realize a complex Business Information System?
- Which Software Development Approach should be choosen to realize a complicated reusable library?

Software Engineering



Engineering is 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.

Software Engineering is the systematic application of engineering knowledge, principles, approaches, practices, methods, and experiences to the development of software.

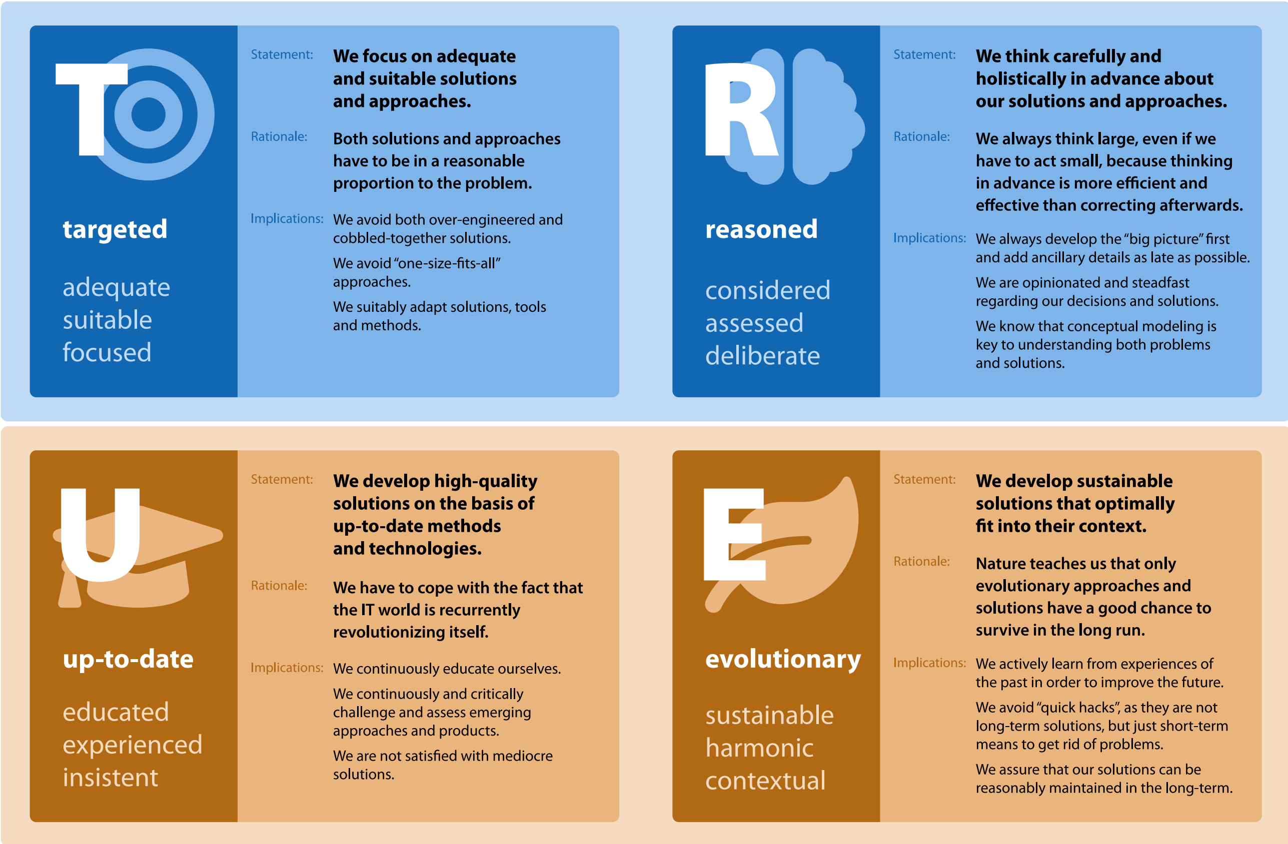
For both Engineering and Software Engineering, the following **Code of Conduct** holds: Commit yourself to the highest ethical and professional conduct; accept responsibility in making decisions consistent with the safety, health, and welfare of the public, apply the state of the art in science and technology of your field; perform services only in areas of your competence; and comply with the principles of sustainable development

Questions

- Is Software Engineering also suitable for the development of a non-complex software in a small team of two people?

TRUE Manifesto

true-manifesto.org



entitlement & values

The TRUE Manifesto states the foundational attitude for good Software Engineering as a central manifesto, similar to the approach – but intentionally in partial contrast with regards content – of the popular Agile Manifesto. The TRUE Manifesto expressed four main aspects.

(T)argeted (adequate, suitable, focused): We focus on adequate and suitable solutions and approaches because: Both solutions and approaches have to be in a reasonable proportion to the problem.

(R)easoned (considered, assessed, deliberate): We think carefully in advance about our planned solutions and approaches because: Thinking is cheaper than correcting.

(U)p-to-date (educated, experienced, insistent): We develop high-quality solutions on the basis of up-to-date methods and technologies because: We have to cope with the fact that the IT world is recurrently revolutionizing itself.

(E)volutionary (sustainable, harmonic, contextual): We develop sustainable solutions that optimally fit into their context because: Nature teaches us that only evolutionary approaches and solutions have a good chance to survive in the long run.

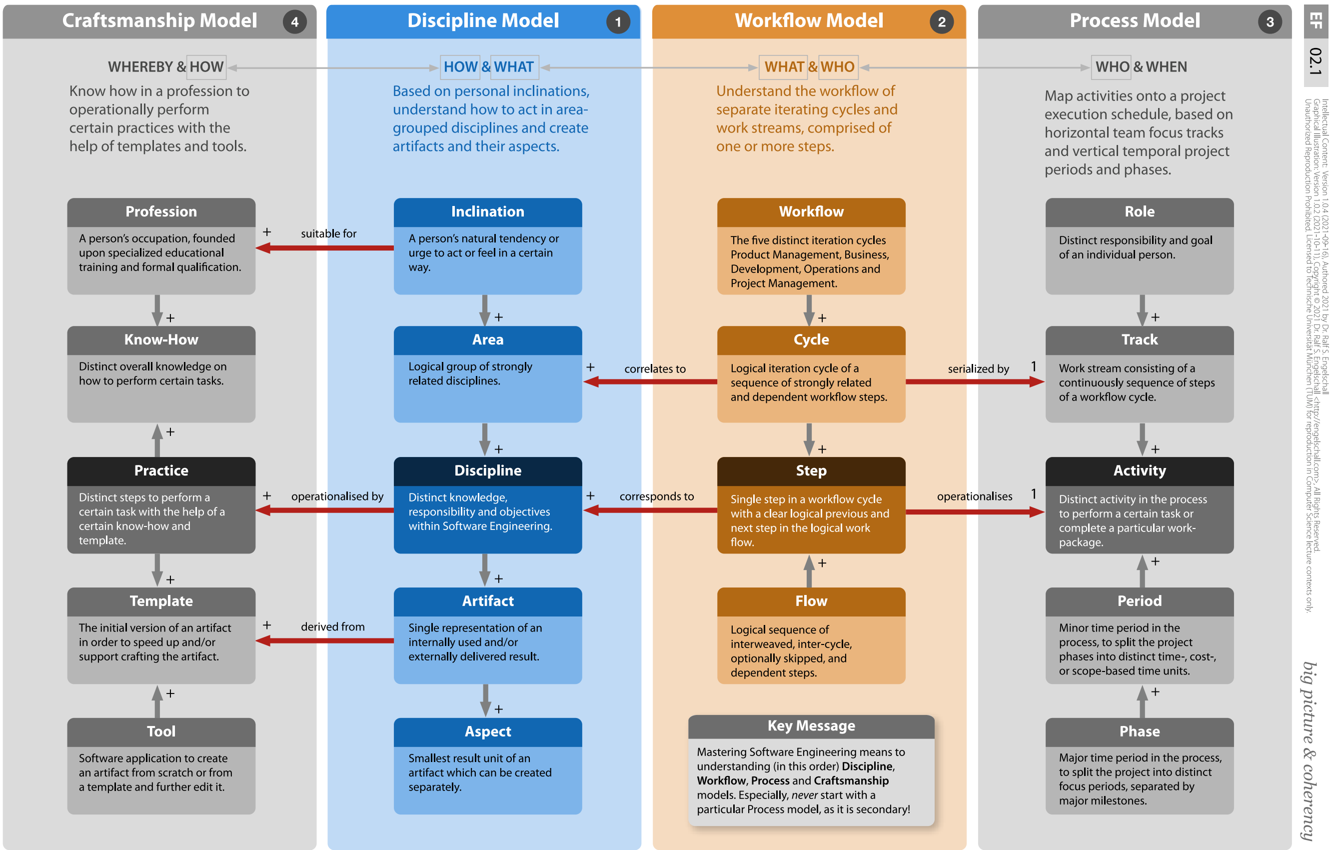
Questions

? Is an extrem agile process with continuous Refactoring in the sense of the TRUE Manifesto for Software Engineering?

ENGINEERING FUNDAMENTALS

Software Engineering Metamodel

TECHNISCHE UNIVERSITÄT MÜNCHEN



Software Engineering can be understood through a meta-model based on four distinct but interlinked models.

The **Craftsmanship Model** is the base and targets the WHEREBY & HOW. It spans from the **Professions** of individual persons, their corresponding **Know-Hows** and **Practices** to the underlying **Templates** and **Tools**.

The **Discipline Model** targets the HOW & WHAT. It segregates Software Engineering into **Disciplines**, which are grouped into **Areas** and which are motivated by the usual **Inclinations** of individual persons. Each Discipline is then described through input and output **Artifacts** and their **Aspects**.

The **Workflow Model** targets the WHAT & WHO. It describes a **Workflow** of **Cycles** which contain **Steps**. A **Flow** are the runs through those Steps over time.

The **Process Model** finally targets the WHO & WHEN. It maps **Activities** onto a project execution schedule, based on horizontal **Tracks** of **Roles** and vertical **Periods** of **Phases**.

Questions

- How many Cycles are known in the Workflow Model of Software Engineering, in which persons with similar Inclinations act?

ENGINEERING FUNDAMENTALS

Software Engineering Disciplines

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 02.2

ANALYSIS (AN)

Software Requirements (REQ) — 3, BB
Identify Needs: We understand which outcomes of the solution are most valuable to users.
Requirements Engineer / Business Analyst

Domain Modeling (DOM) — 2, WB
Determine Solution: We model and specify the solution through involved functional and non-functional aspects.
Business Analyst / Business Architect

ARCHITECTURE (AR)

Software Architecture (SWA) — 1, WB
Design Software: We design an orthogonal, well-balanced and well-considered solution.
Software Architect

System Architecture (SYA) — 2, BB
Design Systems: We ensure that the solution fits optimally into its environment.
System Architect / Enterprise Architect

CONFIGURATION (CF)

Software Versioning (VER) — 1, BB
Version Artifacts: We place every artifact of the solution under strict version control.
Configuration Manager

Software Assembly (ASM) — 1, BB
Assemble Artifacts: We build and package the solution through an automated and repeatable mechanism.
Build Manager / Build Engineer

ANALYTICS (AC)

Software Reviewing (REV) — 4, WB
Review Code: We regularly and semantically peer-review the source code of the solution.
Software Tester

Software Testing (TST) — 2, BB
Test Solution: We adequately test the functional and non-functional aspects of the solution.
Software Tester

MANAGEMENT (MG)

Product Management (PRD) — 3, BB
Push Product: We continuously push the development and release of the solution to the users.
Product Manager / Product Owner

Project Management (PRJ) — 3, BB
Steer Process: We rigorously balance time, cost and scope to react on changes and reach the goals.
Project Manager

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

EXPERIENCE (EX)

User Experience (UXP) — 3, BB
Optimize Workflows: We align the solution to the perspective of the target audience.
User Experience Expert

User Interface (UID) — 2, WB
Design User Interfaces: We design a useful, intuitive, and beautiful user interface for the solution.
User Interface Designer / Graphics Designer

DEVELOPMENT (DV)

Software Development (DEV) — 1, WB
Implement Code: We develop the solution outside-in, from coarse to fine aspects.
Software Engineer / Software Developer

Software Refactoring (REF) — 4, BB
Refactor Code: We regularly and holistically refactor the solution to ensure long-term quality.
Software Engineer / Software Developer

DELIVERY (DL)

Software Deployment (DPL) — 1, BB
Deploy Artifacts: We ship and deploy the solution through an automated and repeatable mechanism.
System Engineer / System Developer

System Operations (OPS) — 4, BB
Operate Solution: We ensure that our infra-structures and the solution can be operated in a resilient and secure way.
System Administrator / System Operator

COMPREHENSION (CP)

Usage Documentation (DOC) — 2, WB
Document Solution: We adequately document the usage and operation of the solution.
Technical Writer

User Training (TRN) — 4, BB
Train Users: We adequately train the users and operators of the solution.
Technical Writer / Product Trainer

ADJUSTMENT (AD)

Project Coaching (COA) — 4, BB
Support Members: We ensure that project members use state-of-the-art methodology, technology, and tools.
Project Coach / Methodology Master

Change Management (CHG) — 3, BB
Involve Stakeholders: We ensure that all stakeholders of the solution are suitably involved.
Change Manager

inclination & knowledge

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

Software Engineering can be understood through 20 distinct **Disciplines** (operationalized through input and output artifacts and their aspects), which are logically grouped into 10 distinct **Areas**, and which in turn are logically grouped into 5 distinct **Inclinations** of individual persons.

Persons with a strong **domain-specific** and **business-oriented** Inclination act in the Areas **Analysis** and **Experience** and the corresponding Disciplines **Software Requirements**, **Domain Modeling**, **User Experience** and **User Interface**.

Persons with a strong **constructive** and **technological** Inclination act in the Areas **Architecture** and **Development** and the corresponding Disciplines **Software Architecture**, **System Architecture**, **Software Development** and **Software Refactoring**.

Persons with a strong **infrastructural** and **technological** Inclination act in the Areas **Configuration** and **Delivery** and the corresponding Disciplines **Software Versioning**, **Software Assembly**, **Software Deployment** and **Software Operations**.

Persons with a strong **analytical** and **domain-specific** Inclination act in the Areas **Analytics** and **Comprehension** and the corresponding Disciplines **Software Reviewing**, **Software Testing**, **Usage Documentation** and **User Training**.

Persons with a strong **people-oriented** and **process-oriented** Inclination act in the Areas **Management** and **Adjustment** and the corresponding Disciplines **Project Management**, **Project Auditing**, **Project Coaching** and **Change Management**.

Questions

❓ Which Disciplines of Software Engineering are considered the **King Disciplines**?