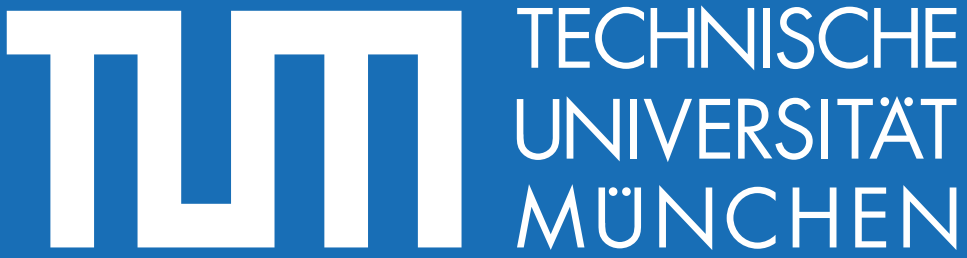
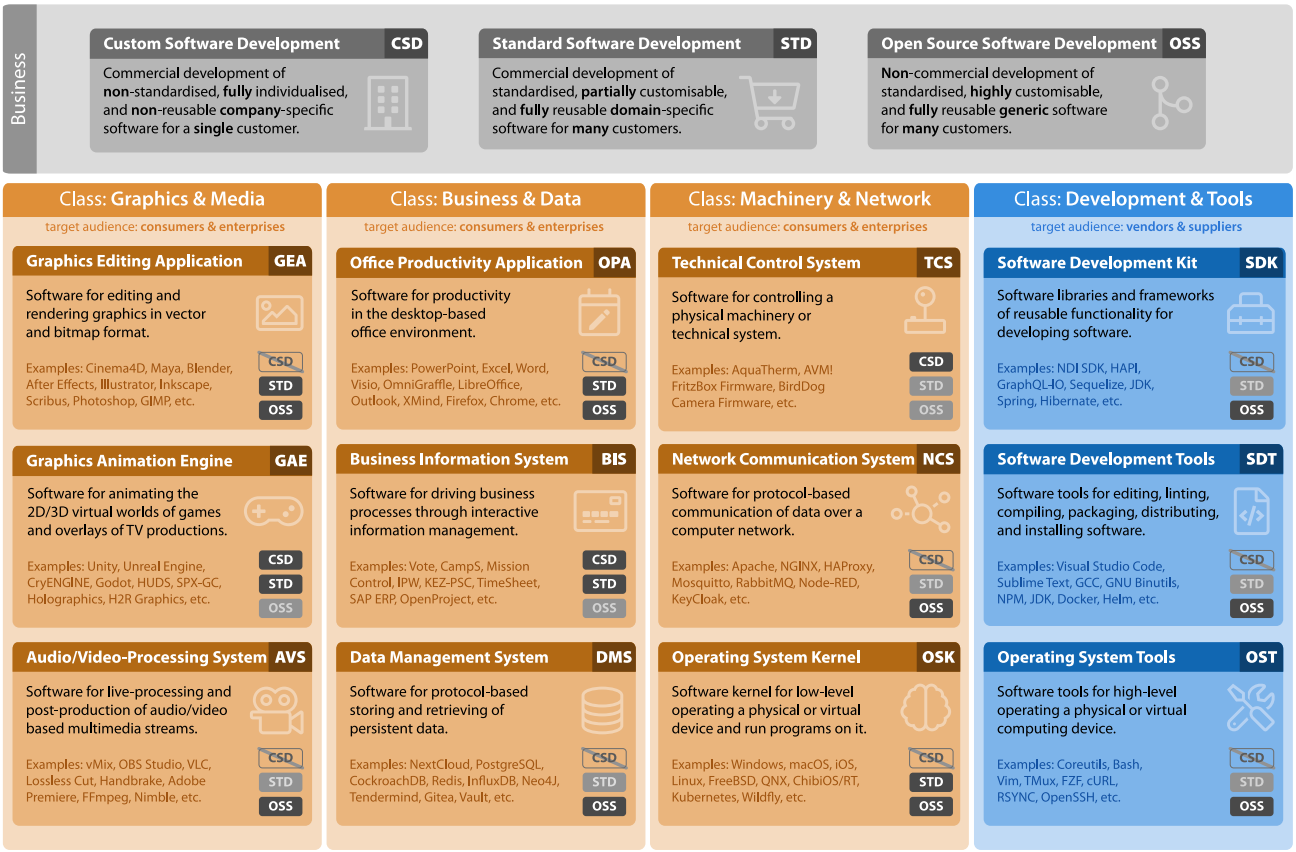


Software Engineering in Industrial Practice (SEIP)

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 many customers; and **Open-Source Software Development**, the non-commercial development of a standardized, highly customizable, and fully reusable technical software for many customers.

There are four areas and 12 classes of software. In the first area, 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.

In the second area, 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.

In the third area, 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 operating a physical or virtual computing device.

In the forth area, 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?

Intellectual Content: Version 1.1.1 (2022-12-10), Authored 2021-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.1.0 (2022-12-10), Copyright © 2021-2022 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

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

Intellectual Content: Version 1.0.4 (2022-11-10), Authored 2021-2022 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.3 (2021-11-05), Copyright © 2021 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

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 production-grade software solution by a professional, clean but plain craftsmanship means in order to solve a usually complicated problem.

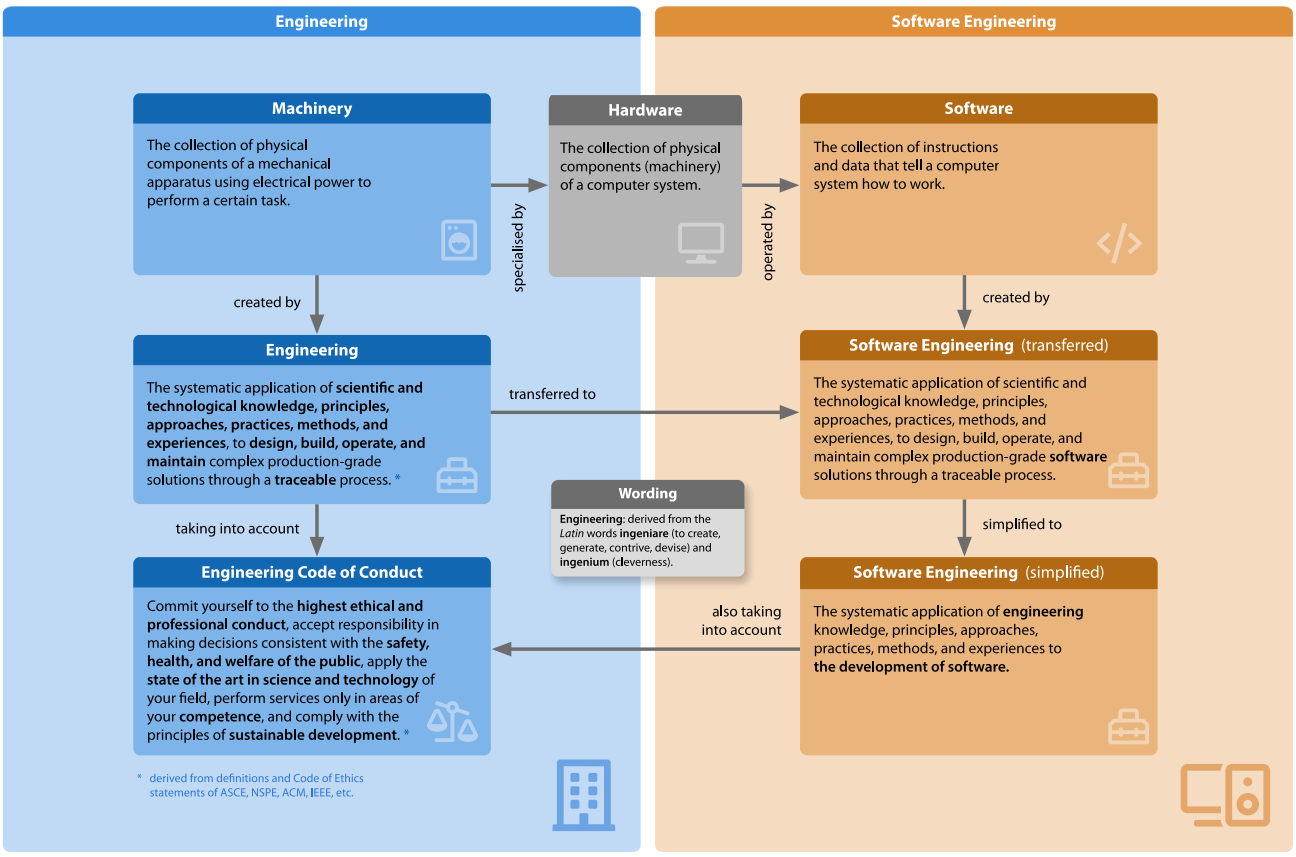
In **Software Engineering**, one develops production-grade software solution by a professional, risk-hedged engineering approach in order to solve a usually complex problem.

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



Intellectual Content: Version 1.0.2 (2021-04-26), Authored 2021 by Dr. Ralf D. Engelken
Graphical Illustration: Version 1.0.2 (2021-04-01), Copyright © 2021 Dr. Ralf D. Engelken (http://engelken.com) – All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contents only.

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.

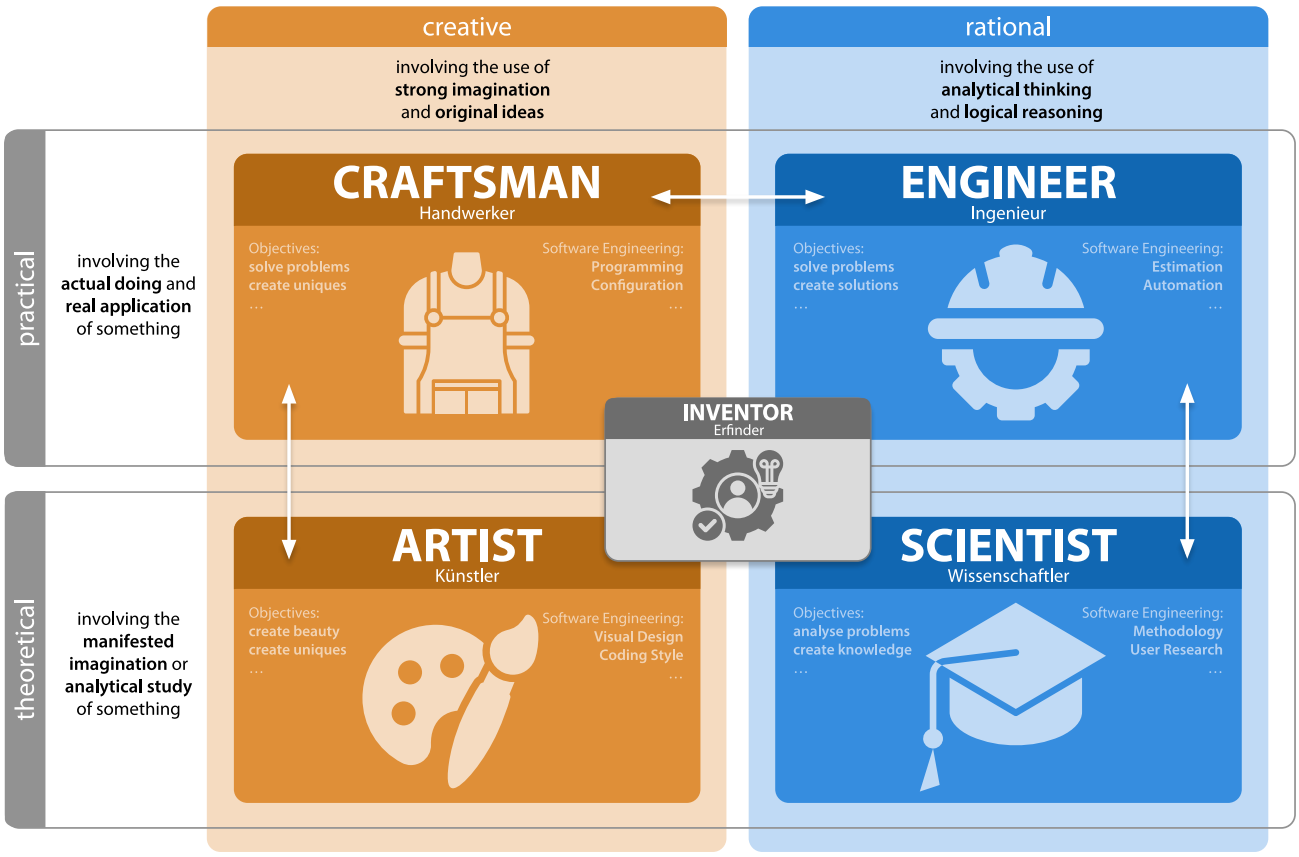
Software Engineering is the systematic application of engineering knowledge, principles, approaches, practices, methods, and experiences to the development of software.

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 the 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?

Profession Characteristics



EF 01.4
Intellectual Content: Version 1.0.0 (2024-04-10) Authored 2024 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.0 (2024-04-10) Copyright © 2024 Dr. Ralf S. Engelschall <http://engelschall.com> All Rights Reserved
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science Lecture contexts only
diagram.direct / XIRF-NIIV

Professions usually have two of four characteristics: Being **creative** means involving the use of strong imagination and original ideas. Being **rational** means involving the use of analytical thinking and logical reasoning. Being **practical** means involving the actual doing and real application of something. Being **theoretical** means involving the manifested imagination or analytical study of something.

One can distinguish five interesting professions: A **craftsman** acts in a creative and practical way, and solves problems and creates uniques. An **engineer** acts in a rational and practical way, and solves problems and creates solutions. An **artist** acts in a creative and theoretical way, and creates beauty and uniques. A **scientist** acts in a rational and theoretical way, and analyses problems and creates knowledge. On the other hand, an **inventor** usually has to combine all characteristics.

Questions

- When you're dealing with configuration and programming in Software Engineering, instead of an engineer you act more like a…?

ENGINEERING FUNDAMENTALS

Discipline Claim

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 01.6

FA — FARSIGHTED
weitblickend

Be farsighted in your solution finding.

Sei weitblickend in deiner Lösungsfindung.

AR: Scalable Hub'n'Spoke
DV: Plugin SPI

TE — TENET-ORIENTED
grundsatzorientiert

Orientate yourself on fixed tenets in your approach and solution finding.

Orientiere dich an festen Grundsätzen in deinem Vorgehen und deiner Lösungsfindung.

AR: Separation of Concern
DV: Strict Coding-Style

TH — THOUGHTFUL
wohlüberlegt

Act thoughtful in your approach and solution finding.

Agiere wohlüberlegt in deinem Vorgehen und deiner Lösungsfindung.

AR: Modularization
DV: Algorithmical Control Structure

HO — HOLISTICALLY
ganzheitlich

Think holistically and in the long-term when finding your solutions.

Denke ganzheitlich und langfristig in deiner Lösungsfindung.

AR: Walking Skeleton Design
DV: Consistent Error Handling

AD — ADEQUATE
angemessen

Ensure that your approach and solutions are adequate to the boundary conditions.

Sorge dafür, daß dein Vorgehen und deine Lösungen angemessen zu den Rahmenbedingungen sind.

AR: No Cloud-Native Complexity
DV: No Over-Engineered Abstractions

FE — FEASIBLE
machbar

Ensure that your approach and solutions can be realised at reasonable costs.

Sorge dafür, daß dein Vorgehen und deine Lösungen mit vernünftigen Kosten realisiert werden können.

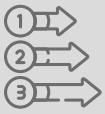
AR: Existing Framework Functionality
DV: Realistic Programming Model

IN — INCREMENTAL
inkrementell

Apply the depth of your discipline incrementally.

Wende die Tiefe deiner Disziplin inkrementell an.

AR: Identified Solution Cruxes
DV: Minimum Viable Product



VA — VALUEABLE
wertvoll

Provide clearly recognizable added values with your approach and solutions.

Liefere klar ersichtliche Mehrwerte mit deinem Vorgehen und deinen Lösungen.

AR: Technology Stack Design
DV: User-Story-Driven Functionality

SU — SUSTAINABLE
nachhaltig

Create sustainable solutions that are well integrated into their environment.

Erschaffe nachhaltige Lösungen, die gut in ihre Umgebung integriert sind.

AR: Interoperable Interfaces
DV: Maintainable Code

Intellectual Content: Version 1.0.1 (2023-06-17), Authored 2023-2024 by Dr. Ralf S. Engelschall
Graphical Illustration: Version 1.0.1 (2024-06-17) Copyright © 2023-2024 Dr. Ralf S. Engelschall <http://engelschall.com>, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contents only.

Across the various Software Engineering **Disciplines**, there are some common properties that they all claim from a conceptual point of view: **farsighted**, **tenet-oriented**, **thoughtful**, **holistically**, **adequate**, **feasible**, **incremental**, **valueable**, and **sustainable**.

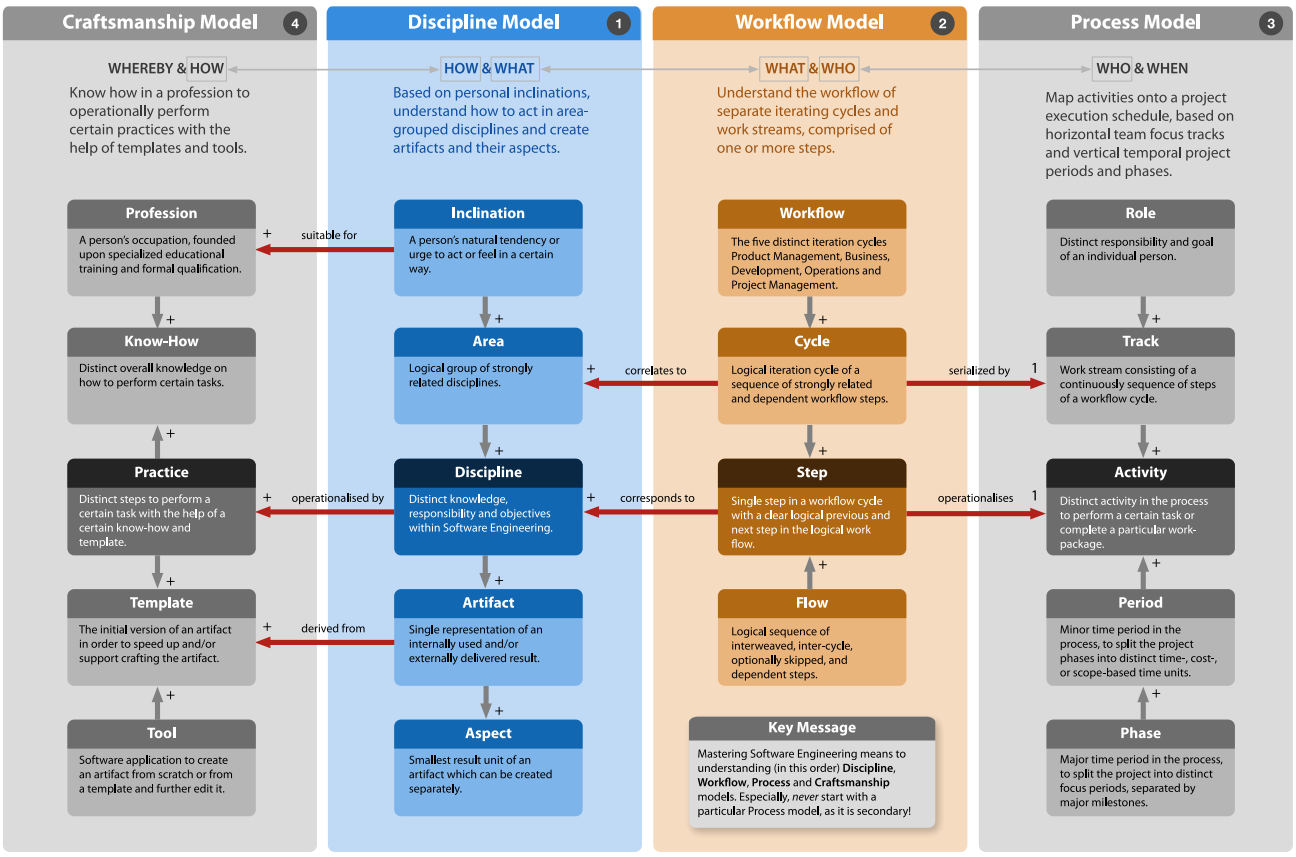
Questions

? What is the most difficult claim in a **Discipline** in today's practice?

ENGINEERING FUNDAMENTALS

Software Engineering Metamodel

TECHNISCHE UNIVERSITÄT MÜNCHEN



EF 02.1

big picture & coherency

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** is the run 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

inclination & knowledge

Intellectual Content: Version 3.0.9 (2021-12-10), Authored 2009-2022 by Dr. Ralf S. Engelschall and Rudolf Koster
Graphical Illustration: Version 3.0.2 (2021-06-11), Copyright © 2009-2021 Dr. Ralf S. Engelschall, http://engelschall.com/, All Rights Reserved.
Unauthorized Reproduction Prohibited. Licensed to Technische Universität München (TUM) for reproduction in Computer Science lecture contexts only.

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**?