TECHNISCHE UNIVERSITÄT MÜNCHEN

Software Engineering in Industrial Practice (SEIP)

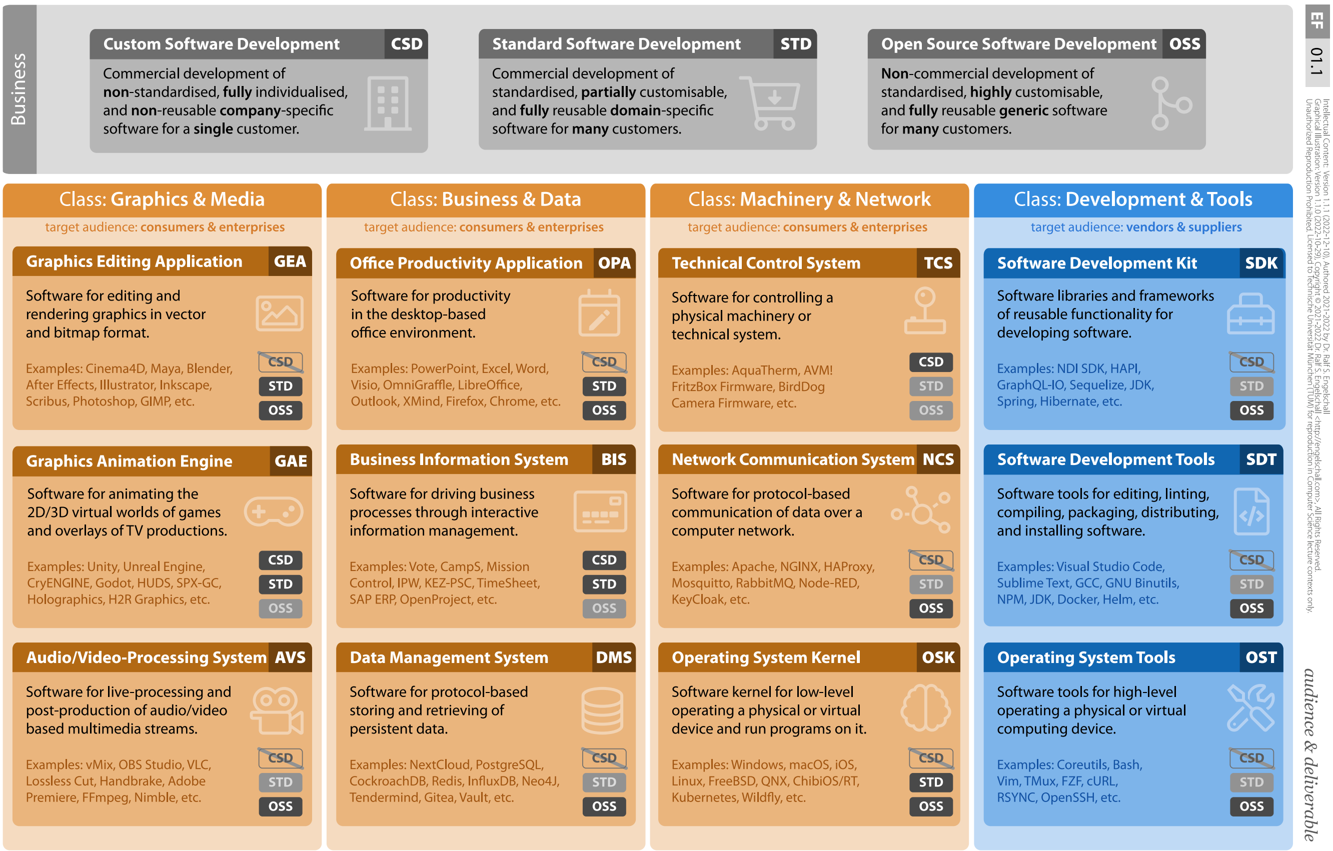
Dr. Ralf S. Engelschall

ENGINEERING FUNDAMENTALS

Software Classes

TECHNISCHE UNIVERSITÄT MÜNCHEN

EF 01.1



Custom Software Development **CSD**
Commercial development of **non**-standardised, **fully** individualised, and **non**-reusable **company**-specific software for a **single** customer.

Standard Software Development **STD**
Commercial development of standardised, **partially** customisable, and **fully** reusable **domain**-specific software for **many** customers.

Open Source Software Development **OSS**
Non-commercial development of standardised, **highly** customisable, and **fully** reusable **generic** software for **many** customers.

Class: **Graphics & Media**
target audience: **consumers & enterprises**

Graphics Editing Application **GEA**
Software for editing and rendering graphics in vector and bitmap format.
Examples: Cinema4D, Maya, Blender, After Effects, Illustrator, Inkscape, Scribus, Photoshop, GIMP, etc.
~~CSD~~ STD OSS

Graphics Animation Engine **GAE**
Software for animating the 2D/3D virtual worlds of games and overlays of TV productions.
Examples: Unity, Unreal Engine, CryENGINE, Godot, HUDS, SPX-GC, Holographics, H2R Graphics, etc.
CSD STD OSS

Audio/Video-Processing System **AVS**
Software for live-processing and post-production of audio/video based multimedia streams.
Examples: vMix, OBS Studio, VLC, Lossless Cut, Handbrake, Adobe Premiere, FFmpeg, Nimble, etc.
~~CSD~~ STD OSS

Class: **Business & Data**
target audience: **consumers & enterprises**

Office Productivity Application **OPA**
Software for productivity in the desktop-based office environment.
Examples: PowerPoint, Excel, Word, Visio, OmniGraffle, LibreOffice, Outlook, XMind, Firefox, Chrome, etc.
~~CSD~~ STD OSS

Business Information System **BIS**
Software for driving business processes through interactive information management.
Examples: Vote, CampS, Mission Control, IPW, KEZ-PSC, TimeSheet, SAP ERP, OpenProject, etc.
CSD STD OSS

Data Management System **DMS**
Software for protocol-based storing and retrieving of persistent data.
Examples: NextCloud, PostgreSQL, CockroachDB, Redis, InfluxDB, Neo4J, Tendermind, Gitea, Vault, etc.
~~CSD~~ STD OSS

Class: **Machinery & Network**
target audience: **consumers & enterprises**

Technical Control System **TCS**
Software for controlling a physical machinery or technical system.
Examples: AquaTherm, AVM! FritzBox Firmware, BirdDog Camera Firmware, etc.
CSD STD OSS

Network Communication System **NCS**
Software for protocol-based communication of data over a computer network.
Examples: Apache, NGINX, HAProxy, Mosquitto, RabbitMQ, Node-RED, KeyCloak, etc.
~~CSD~~ STD OSS

Operating System Kernel **OSK**
Software kernel for low-level operating a physical or virtual device and run programs on it.
Examples: Windows, macOS, iOS, Linux, FreeBSD, QNX, ChibiOS/RT, Kubernetes, Wildfly, etc.
~~CSD~~ STD OSS

Class: **Development & Tools**
target audience: **vendors & suppliers**

Software Development Kit **SDK**
Software libraries and frameworks of reusable functionality for developing software.
Examples: NDI SDK, HAPI, GraphQL-IO, Sequelize, JDK, Spring, Hibernate, etc.
~~CSD~~ STD OSS

Software Development Tools **SDT**
Software tools for editing, linting, compiling, packaging, distributing, and installing software.
Examples: Visual Studio Code, Sublime Text, GCC, GNU Binutils, NPM, JDK, Docker, Helm, etc.
~~CSD~~ STD OSS

Operating System Tools **OST**
Software tools for high-level operating a physical or virtual computing device.
Examples: Coreutils, Bash, Vim, TMux, FZF, cURL, RSYNC, OpenSSH, etc.
~~CSD~~ STD OSS

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.

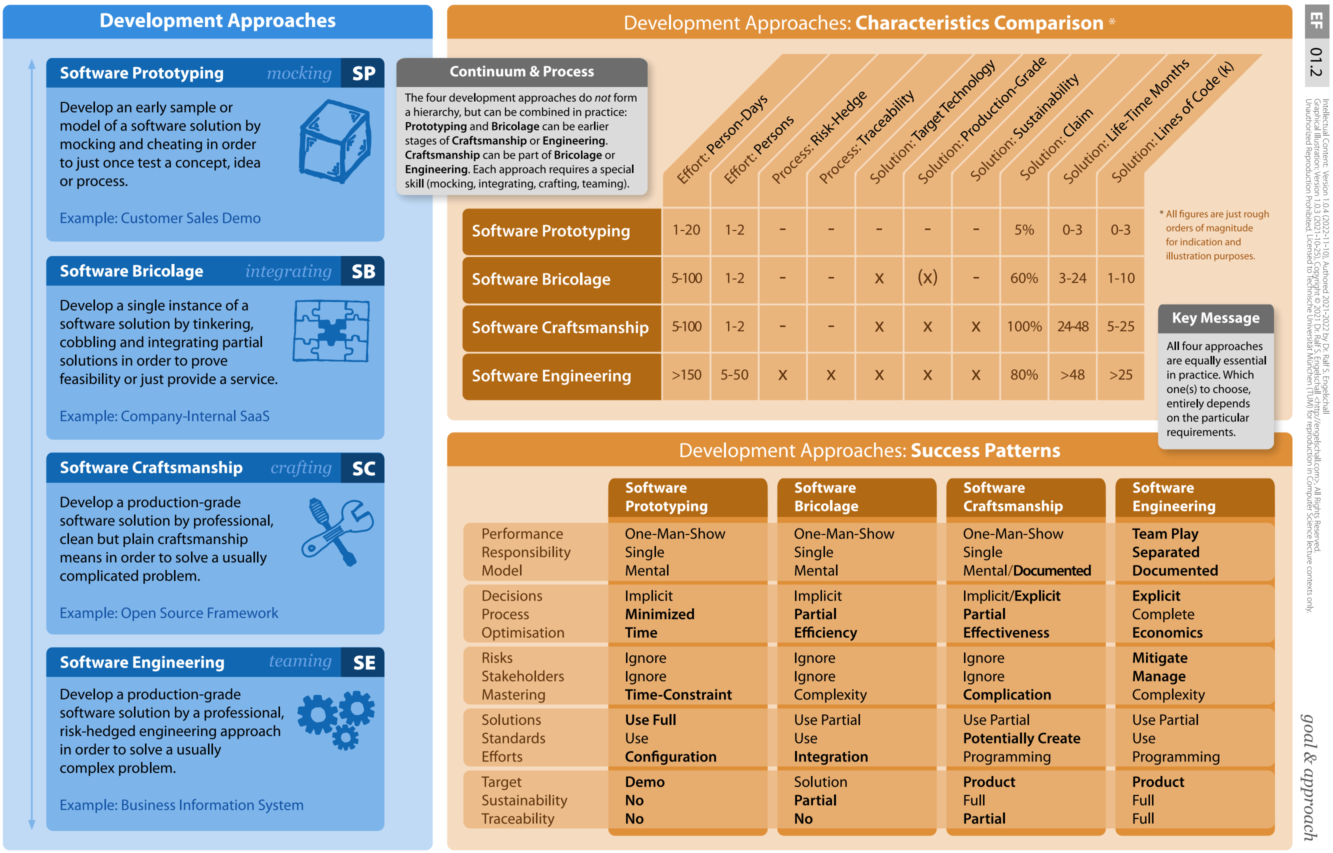
Questions

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

ENGINEERING FUNDAMENTALS

Software Development Approaches

TECHNISCHE UNIVERSITÄT MÜNCHEN



EF 01.2

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

Development Approaches: **Characteristics Comparison** *

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).

	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 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.

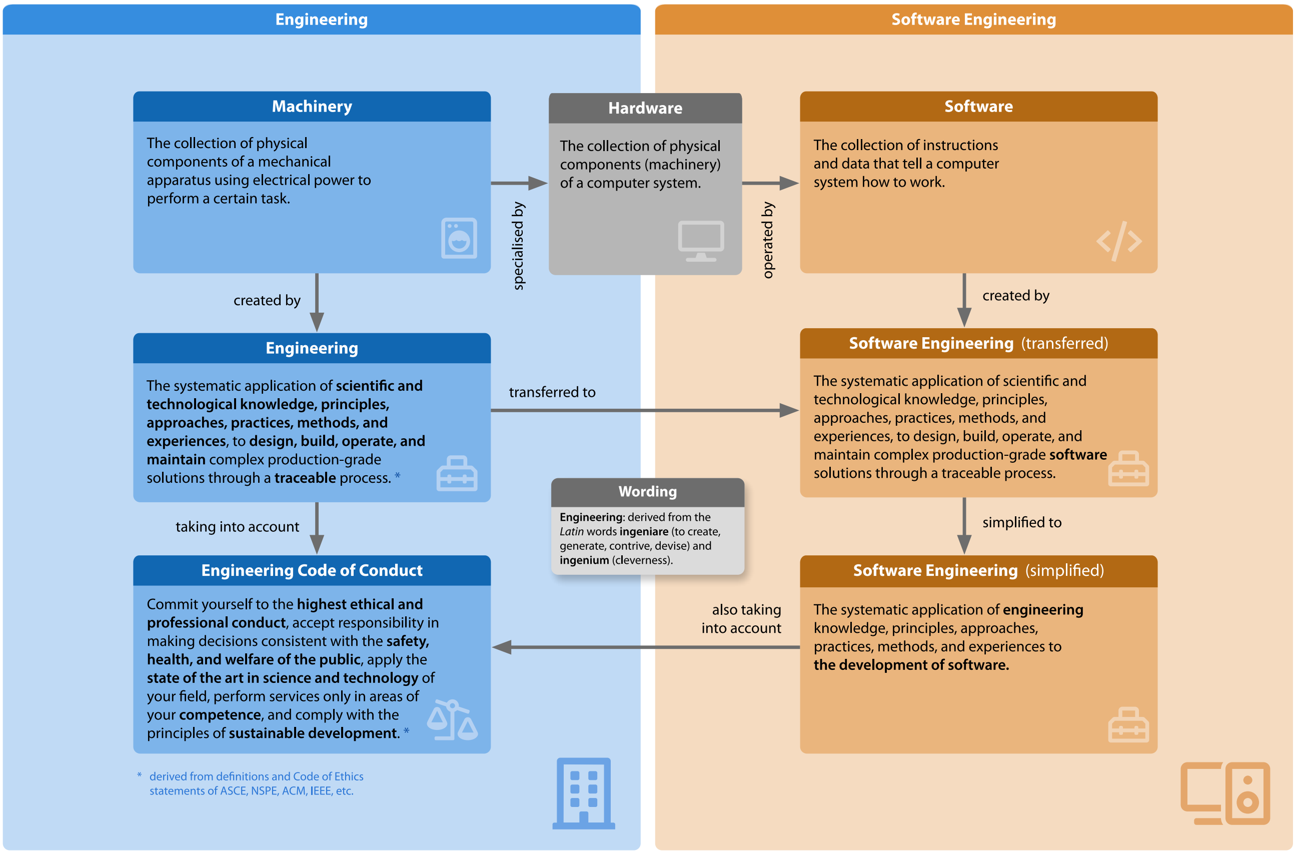
Questions

- ? 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?

ENGINEERING FUNDAMENTALS

Software Engineering

TECHNISCHE UNIVERSITÄT MÜNCHEN



Engineering

Machinery

The collection of physical components of a mechanical apparatus using electrical power to perform a certain task.

Engineering

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. *

Engineering Code of Conduct

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**. *

* derived from definitions and Code of Ethics statements of ASCE, NSPE, ACM, IEEE, etc.

Hardware

The collection of physical components (machinery) of a computer system.

Wording

Engineering: derived from the *Latin* words **ingeniare** (to create, generate, contrive, devise) and **ingenium** (cleverness).

Software Engineering

Software

The collection of instructions and data that tell a computer system how to work.

Software Engineering (transferred)

The systematic application of scientific and technological knowledge, principles, approaches, practices, methods, and experiences, to design, build, operate, and maintain complex production-grade **software** solutions through a traceable process.

Software Engineering (simplified)

The systematic application of **engineering** knowledge, principles, approaches, practices, methods, and experiences to **the development of software.**

profession & ethics

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.

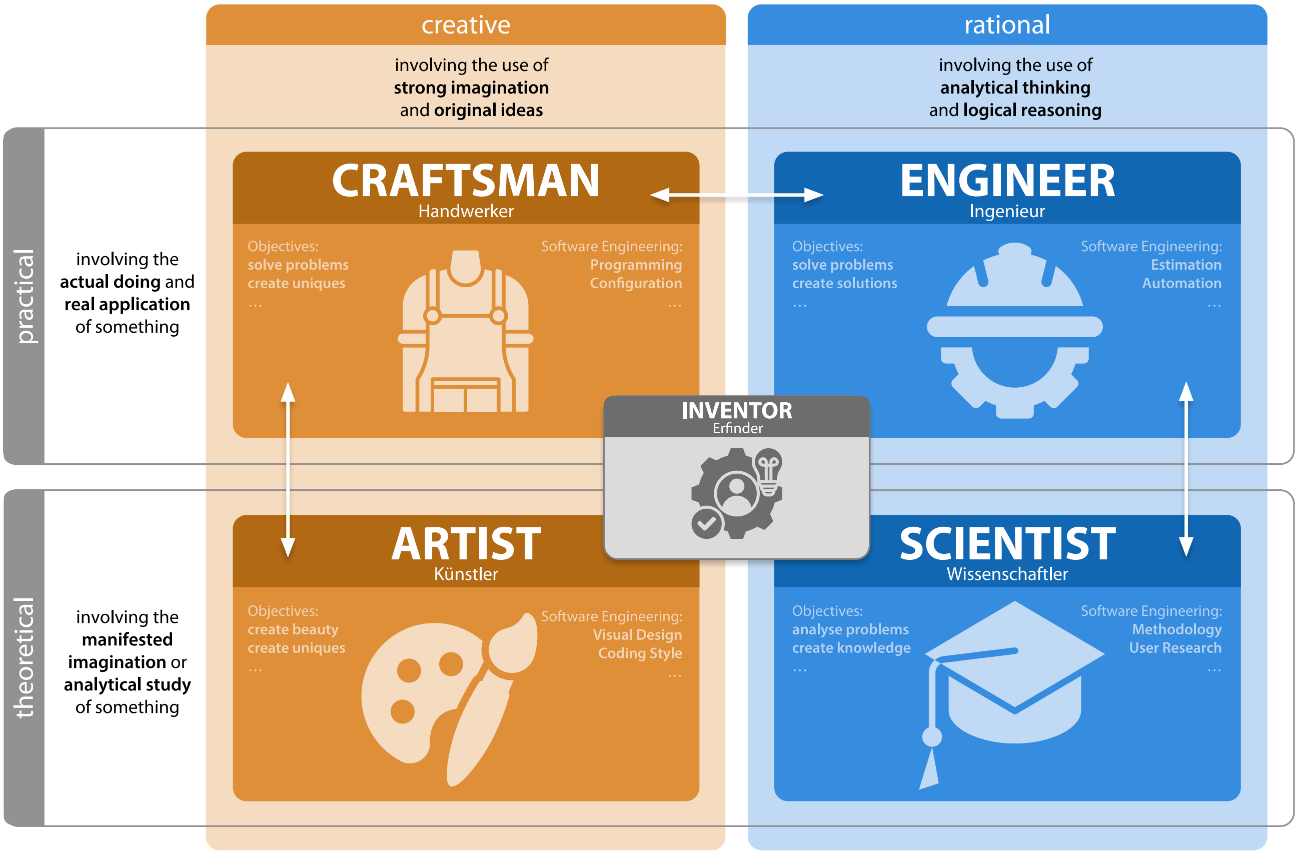
Questions

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

ENGINEERING FUNDAMENTALS

Profession Characteristics

TECHNISCHE UNIVERSITÄT MÜNCHEN



creative

involving the use of **strong imagination** and **original ideas**

rational

involving the use of **analytical thinking** and **logical reasoning**

practical

involving the **actual doing** and **real application** of something

theoretical

involving the **manifested imagination** or **analytical study** of something

CRAFTSMAN
Handwerker

Objectives:
solve problems
create uniques
…

Software Engineering:
Programming
Configuration
…

ENGINEER
Ingenieur

Objectives:
solve problems
create solutions
…

Software Engineering:
Estimation
Automation
…

INVENTOR
Erfinder

ARTIST
Künstler

Objectives:
create beauty
create uniques
…

Software Engineering:
Visual Design
Coding Style
…

SCIENTIST
Wissenschaftler

Objectives:
analyse problems
create knowledge
…

Software Engineering:
Methodology
User Research
…

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…?

EF 01.4

diagram.direct / XIRF-NIIV

Discipline Claim

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

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**.

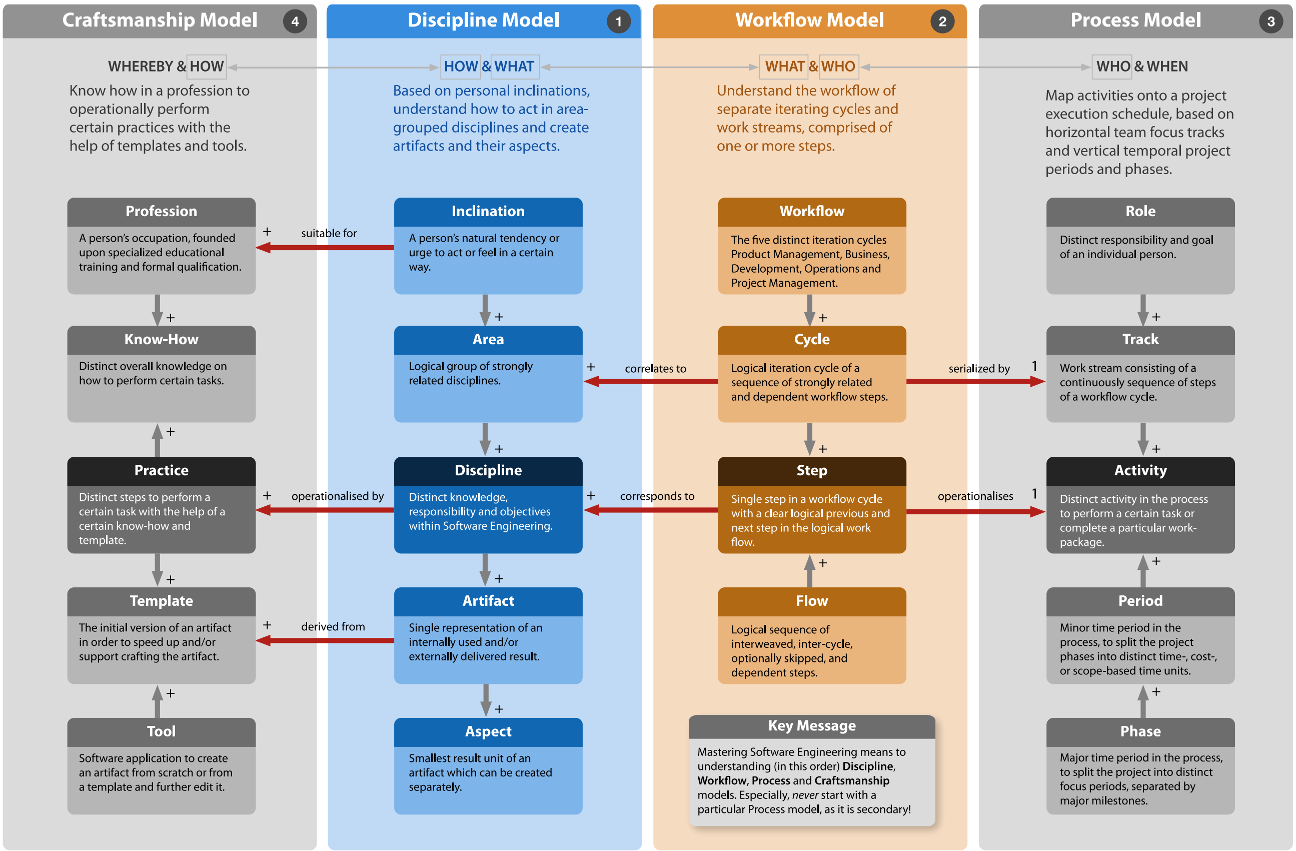
Questions

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

ENGINEERING FUNDAMENTALS

Software Engineering Metamodel

TECHNISCHE UNIVERSITÄT MÜNCHEN



Craftsmanship Model 4

WHEREBY & HOW

Know how in a profession to operationally perform certain practices with the help of templates and tools.

Profession: A person's occupation, founded upon specialized educational training and formal qualification.

Know-How: Distinct overall knowledge on how to perform certain tasks.

Practice: Distinct steps to perform a certain task with the help of a certain know-how and template.

Template: The initial version of an artifact in order to speed up and/or support crafting the artifact.

Tool: Software application to create an artifact from scratch or from a template and further edit it.

Discipline Model 1

HOW & WHAT

Based on personal inclinations, understand how to act in area-grouped disciplines and create artifacts and their aspects.

Inclination: A person's natural tendency or urge to act or feel in a certain way.

Area: Logical group of strongly related disciplines.

Discipline: Distinct knowledge, responsibility and objectives within Software Engineering.

Artifact: Single representation of an internally used and/or externally delivered result.

Aspect: Smallest result unit of an artifact which can be created separately.

Workflow Model 2

WHAT & WHO

Understand the workflow of separate iterating cycles and work streams, comprised of one or more steps.

Workflow: The five distinct iteration cycles Product Management, Business, Development, Operations and Project Management.

Cycle: Logical iteration cycle of a sequence of strongly related and dependent workflow steps.

Step: Single step in a workflow cycle with a clear logical previous and next step in the logical work flow.

Flow: Logical sequence of interweaved, inter-cycle, optionally skipped, and dependent steps.

Key Message: Mastering Software Engineering means to understanding (in this order) **Discipline**, **Workflow**, **Process** and **Craftsmanship** models. Especially, *never* start with a particular Process model, as it is secondary!

Process Model 3

WHO & WHEN

Map activities onto a project execution schedule, based on horizontal team focus tracks and vertical temporal project periods and phases.

Role: Distinct responsibility and goal of an individual person.

Track: Work stream consisting of a continuously sequence of steps of a workflow cycle.

Activity: Distinct activity in the process to perform a certain task or complete a particular work-package.

Period: Minor time period in the process, to split the project phases into distinct time-, cost-, or scope-based time units.

Phase: Major time period in the process, to split the project into distinct focus periods, separated by major milestones.

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**.

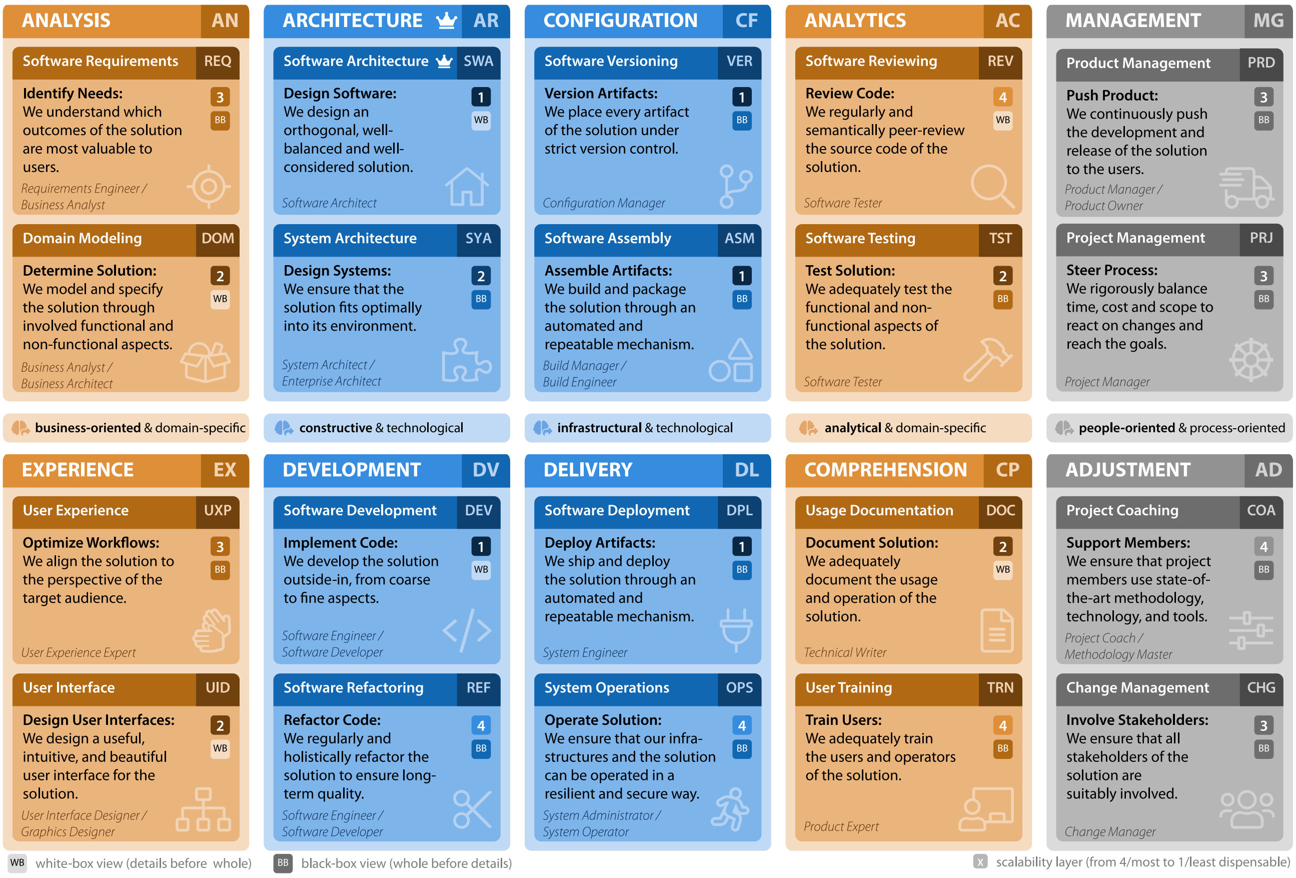
Questions

? 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



ANALYSIS AN

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

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

ARCHITECTURE ♛ AR

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

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

CONFIGURATION CF

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

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

ANALYTICS AC

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

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

MANAGEMENT MG

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

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

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

EXPERIENCE EX

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

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

DEVELOPMENT DV

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

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

DELIVERY DL

Software Deployment DPL
Deploy Artifacts: We ship and deploy the solution through an automated and repeatable mechanism. 1 BB
System Engineer

System Operations OPS
Operate Solution: We ensure that our infrastructures and the solution can be operated in a resilient and secure way. 4 BB
System Administrator / System Operator

COMPREHENSION CP

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

User Training TRN
Train Users: We adequately train the users and operators of the solution. 4 BB
Product Expert

ADJUSTMENT AD

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

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

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

inclination & knowledge

EF 02.2

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**?