



Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall

Business

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 a **class of customers**.

Open Source Software Development

OSS

Non-commercial development of standardised, **highly** customisable, and **fully reusable generic** software for a **class of 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.

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, HUGO, SPX-GC, Holographics, H2R Graphics, etc.

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, Nimbble, etc.

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, Outlook, XMind, Firefox, Chrome, etc.

Business Information System

BIS

Software for driving business processes through interactive information management.

Examples: Voto, Camp5, Mission Control, IPW, KEZ-PSC, TimeSheet, SAP ERP, OpenProject, etc.

Data Management System

DMS

Software for protocol-based storing and retrieving of persistent data.

Examples: NextCloud, PostgreSQL, CockroachDB, Redis, InfluxDB, Neo4J, Tendermint, Gitea, Vault, etc.

Class: Machinery & Network

target audience: consumers & enterprises

Technical Control System

TCS

Software for controlling a physical machinery or technical system.

Examples: AquaTherm, AVMI FritzBox Firmware, BirdDog Camera Firmware, etc.

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.

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.

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-HQ, Sequelize, JDK, Spring, Hibernate, etc.

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.

Operating System Tools

OST

Software tools for high-level operating a physical or virtual computing device.

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.

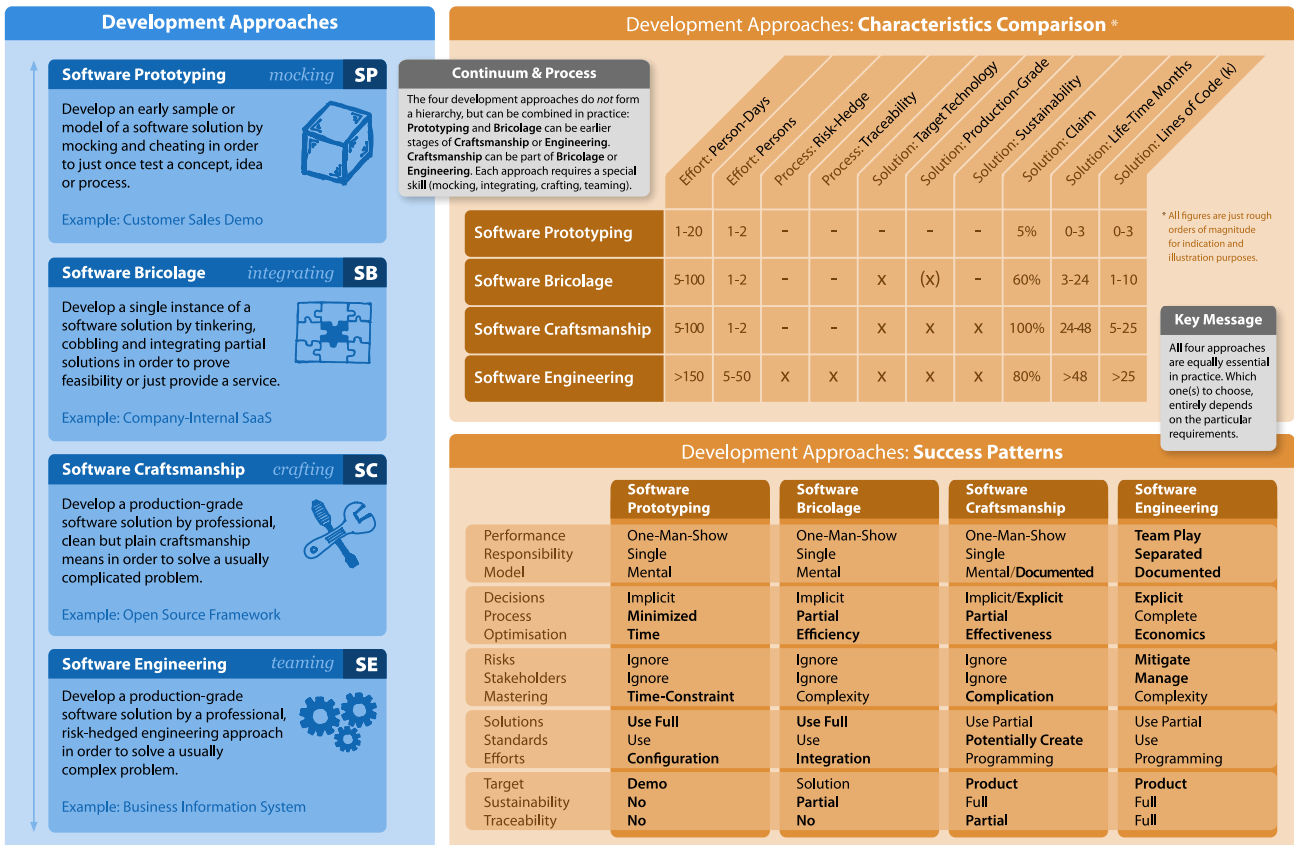
There are three classes of software focusing on Business & Data: **Office Productivity Application**, software for productivity in the desktop-based office environment; **Business Information System**, software for driving business processes through information management; and **Data Management System**,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



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.

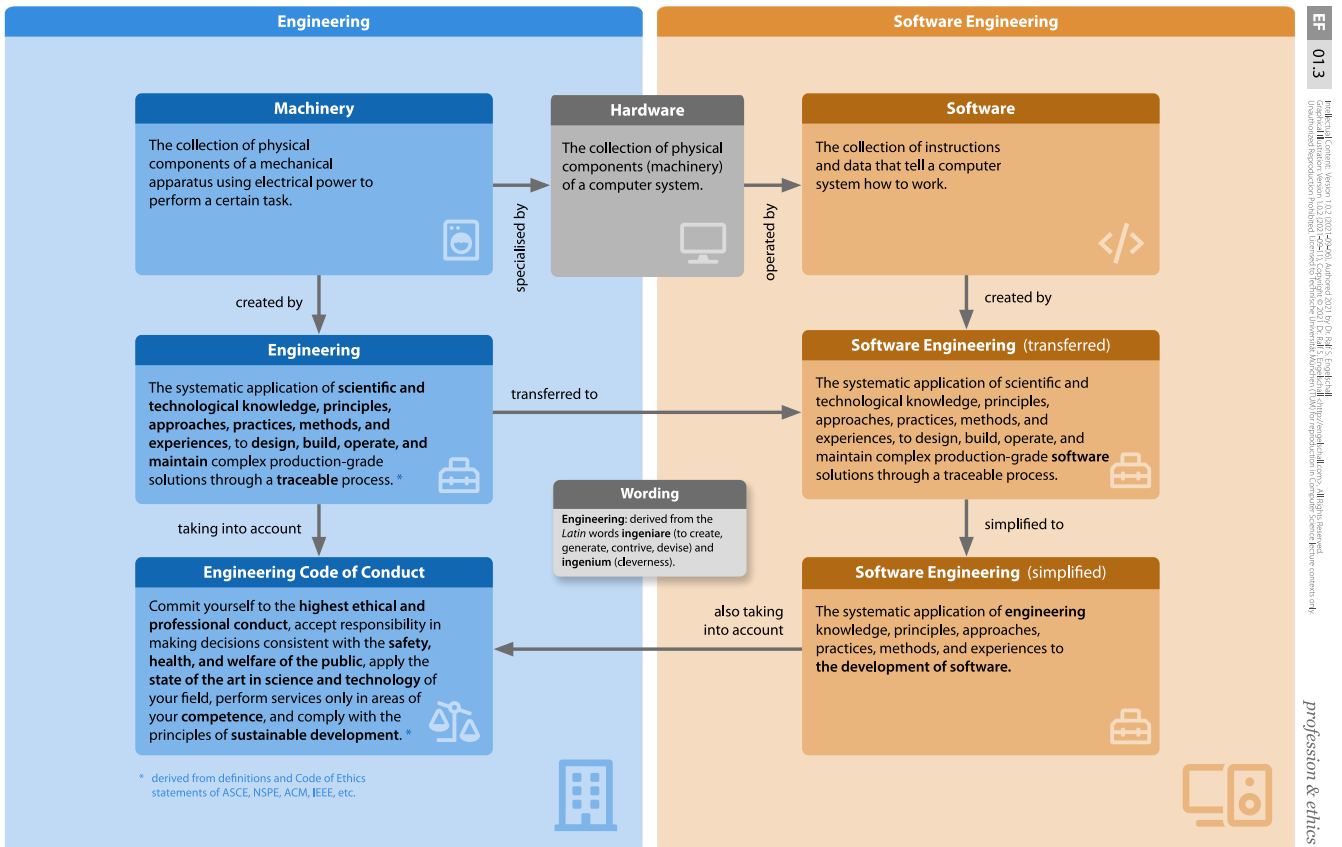
In **Software Craftsmanship**, one develops a production-grade software solution by professional, clean but plain craftsmanship means in order to solve a usually complicated problem.

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




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?

 <p>targeted</p> <p>adequate suitable focused</p>	<p>Statement: We focus on adequate and suitable solutions and approaches.</p> <p>Rationale: Both solutions and approaches have to be in a reasonable proportion to the problem.</p> <p>Implications: We avoid both over-engineered and cobbled-together solutions. We avoid "one-size-fits-all" approaches. We suitably adapt solutions, tools and methods.</p>	 <p>reasoned</p> <p>considered assessed deliberate</p>	<p>Statement: We think carefully and holistically in advance about our solutions and approaches.</p> <p>Rationale: We always think large, even if we have to act small, because thinking in advance is more efficient and effective than correcting afterwards.</p> <p>Implications: We always develop the "big picture" first and add ancillary details as late as possible. We are opinionated and steadfast regarding our decisions and solutions. We know that conceptual modeling is key to understanding both problems and solutions.</p>
 <p>up-to-date</p> <p>educated experienced insistent</p>	<p>Statement: We develop high-quality solutions on the basis of up-to-date methods and technologies.</p> <p>Rationale: We have to cope with the fact that the IT world is recurrently revolutionizing itself.</p> <p>Implications: We continuously educate ourselves. We continuously and critically challenge and assess emerging approaches and products. We are not satisfied with mediocre solutions.</p>	 <p>evolutionary</p> <p>sustainable harmonic contextual</p>	<p>Statement: We develop sustainable solutions that optimally fit into their context.</p> <p>Rationale: Nature teaches us that only evolutionary approaches and solutions have a good chance to survive in the long run.</p> <p>Implications: We actively learn from experiences of the past in order to improve the future. We avoid "quick hacks", as they are not long-term solutions, but just short-term means to get rid of problems. We assure that our solutions can be reasonably maintained in the long-term.</p>

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.

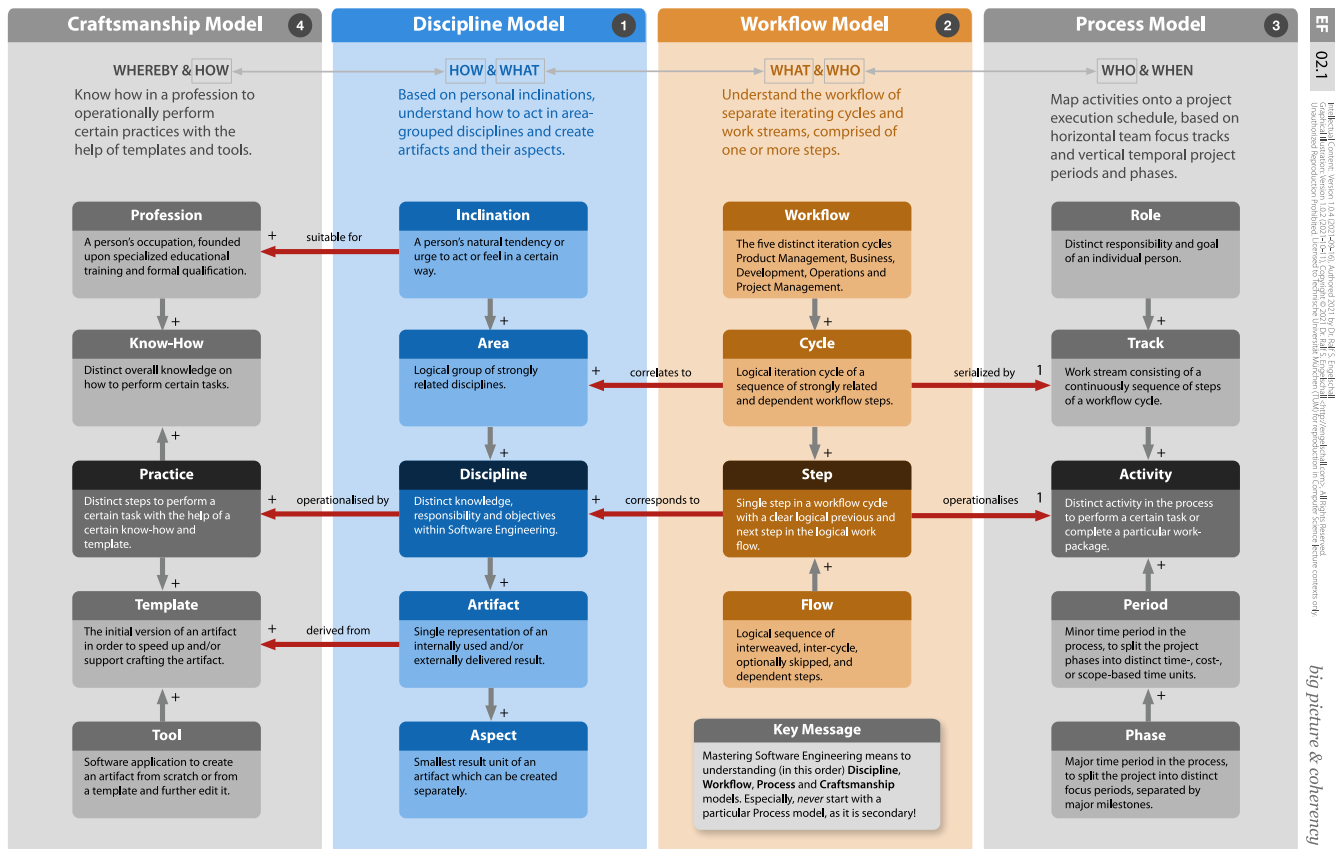
(R)easoned (considered, assessed, deliberate): We think carefully in advance about our planned solutions and approaches because: Thinking is cheaper than correcting.

(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)voluntary (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?



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

The **Craftmanship 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?



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**?