



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 **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.

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, Nimbale, 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, LibreOffice, Outlook, XMind, Firefox, Chrome, etc.

Business Information System

BIS

Software for driving business processes through interactive information management.

Examples: VOTE, 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.

Examples: Coreutils, Bash, Vim, TMux, FZF, cURL, RSYNC, OpenSSH, etc.

Es gibt drei traditionelle Ansätze der Software-Entwicklung: **Custom Software Development**, die kommerzielle Entwicklung von nicht-standardisierter, vollständig individualisierter und nicht wiederverwendbarer firmenspezifischer Software für einen einzelnen Kunden; **Standard Software Development**, die kommerzielle Entwicklung einer standardisierten, teilweise individualisierbaren, aber vollständig wiederverwendbaren fachlichen Software für viele Kunden; und **Open-Source-Software Development**, die nicht-kommerzielle Entwicklung einer standardisierter, hochgradig anpassbarer und vollständig wiederverwendbarer technischen Software für viele Kunden.

Es gibt vier Bereiche und 12 Klassen von Software. Im ersten Bereich gibt es drei Klassen von Software, die sich auf Grafik & Medien fokussieren: **Graphics Editing Application**, Software zum Bearbeiten und Rendern von Grafiken im Vektor- und Bitmap-Format; **Graphics Animation Engine**, Software für die Animation der virtuellen 2D/3D-Welten von Spielen und der Overlays von TV-Produktionen; und **Audio/Video-Processing Systems**, Software für die Live-Bearbeitung und Nachbearbeitung von Audio/Video-Multimedia-Streams.

Im zweiten Bereich gibt es drei Klassen von Software, die sich auf Geschäft & Daten fokussieren: **Office Productivity Application**, eine Software für Produktivität in der Desktop-basierten Büroumgebung; **Business Information System**, eine Software zur Steuerung von Geschäftsprozessen mittels Informationsmanagement; und **Data Management System**, eine Software zum Speichern und Abrufen von persistenten Daten.

Im dritten Bereich gibt es drei Klassen von Software, die sich auf Maschinen & Netzwerk fokussieren: **Technical Control System**, eine Software zur Steuerung einer physischen Maschine oder eines technischen Systems; **Network Communication System**, eine Software für die Kommunikation von Daten über ein Computer-Netzwerk; und **Operating System Kernel**, ein Software-Kern für den Betrieb eines physischen oder virtuellen Computers auf einer niedrigeren Ebene.

Im vierten Bereich gibt es drei Klassen von Software, die sich auf Entwicklung & Werkzeuge fokussieren: **Software Development Kit**, Software-Bibliotheken und -Frameworks mit wiederverwendbarer Funktionalität für die Entwicklung von Software; **Software Development Tools**, Software-Werkzeuge zum Bearbeiten, Analysieren, Übersetzen, Paketieren und Installieren von Software; und **Operating System Tools**, Software-Werkzeuge für den Betrieb eines physischen oder virtuellen Computers.

Fragen

- Welche beiden Klassen von Software werden vor allem mit Hilfe des Ansatzes **Custom Software Development** entwickelt?

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).

	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	100%	24-48	5-25	
Software Engineering	>150	5-50	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

Man kann vier Arten von Software-Entwicklungs-Ansätzen unterscheiden.

Bei **Software Prototyping** entwickelt man ein frühes Beispiel oder Modell einer Softwarelösung durch "Mocking" und "Cheating", um ein Konzept, eine Idee oder einen Prozess einmalig zu testen.

Bei **Software Bricolage** entwickelt man eine einzelne Instanz einer Softwarelösung durch Basteln, Zusammenschustern und Integrieren von Teillösungen, um eine Machbarkeit zu beweisen oder einfach nur eine Dienstleistung zu erbringen.

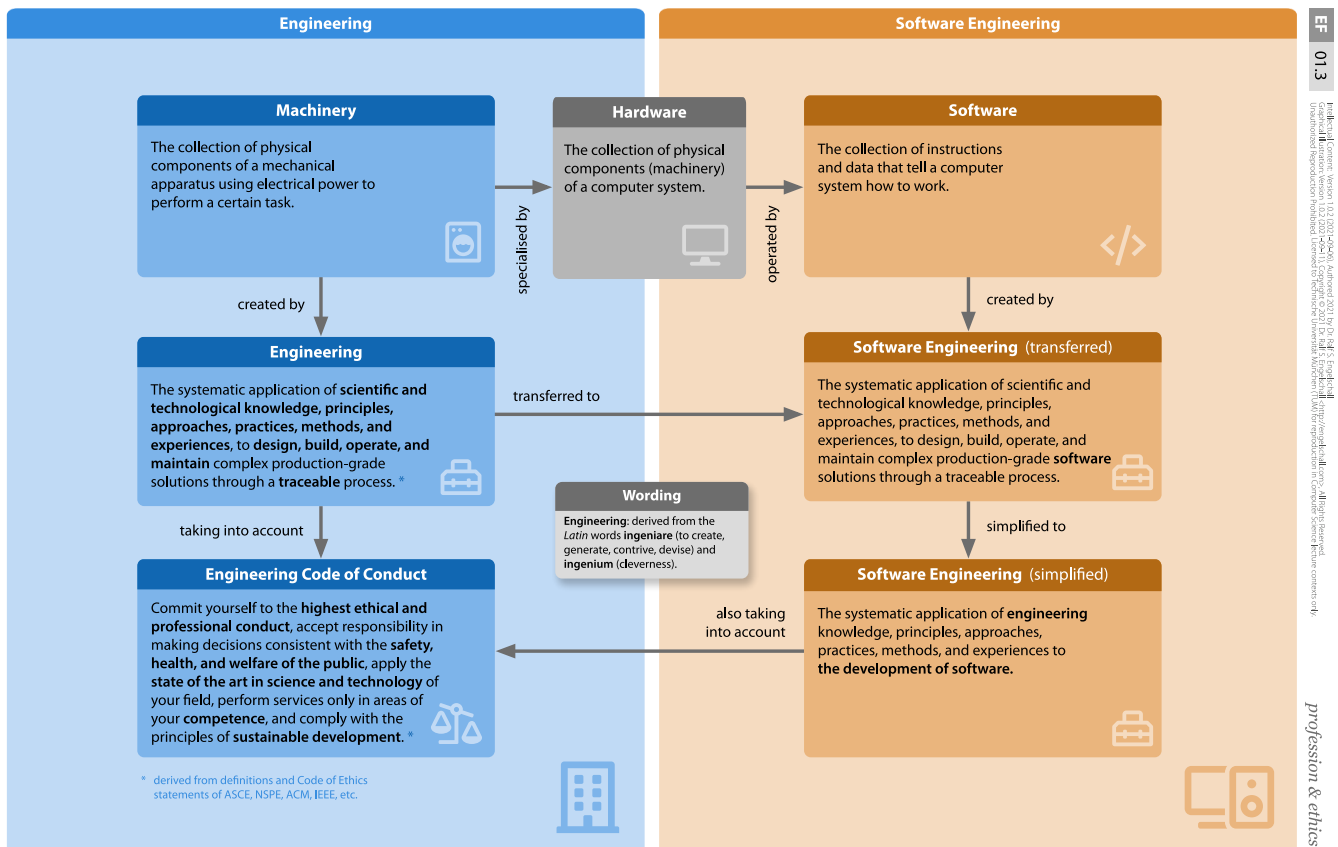
Bei **Software Craftsmanship** entwickelt man eine produktionsreife Softwarelösung mit professionellen, sauberen, aber reinen handwerklichen Mitteln, um ein meist kompliziertes Problem zu lösen.

Bei **Software Engineering** entwickelt man eine produktionsreife Softwarelösung mit einem professionellen, risikoabsichernden Ingenieurs-Ansatz, um ein in der Regel komplexes Problem zu lösen.

Die vier Entwicklungsansätze können in der Praxis auch kombiniert werden: Prototyping und Bricolage können Vorstufen von Craftsmanship oder Engineering sein. Craftsmanship kann Teil des Engineering sein. Jeder Ansatz erfordert eine spezielle Fähigkeit. Alle vier Ansätze sind in der Praxis gleichermaßen wichtig. Welche(s) man jeweils wählt, hängt ganz von den konkreten Anforderungen ab.

Fragen

- Welchen Software-Entwicklungs-Ansatz sollte man wählen, um ein komplexes betriebliches Informationssystem zu realisieren?
- Welchen Software-Entwicklungs-Ansatz sollte man wählen, um eine komplizierte wiederverwendbare Bibliothek zu realisieren?



Ingenieurwesen (Engineering) ist die systematische Anwendung von wissenschaftlichen und technologischen Wissen, Prinzipien, Ansätze, Praktiken, Methoden und Erfahrungen, um komplexe, produktionsreife Lösungen durch einen nachvollziehbaren Prozess zu entwerfen, zu bauen, zu betreiben und zu warten.

Software Engineering ist die systematische Anwendung von ingenieurwissenschaftlichen Wissen, Prinzipien, Ansätzen, Praktiken, Methoden und Erfahrungen auf die Entwicklung von Software.

Sowohl für das Engineering als auch für das Software Engineering gilt der folgende **Verhaltenskodex** (Code of Conduct): Verpflichten Sie sich zu höchstem ethischen und professionellen Verhalten; übernehmen Sie Verantwortung Entscheidungen im Einklang mit der Sicherheit, der Gesundheit und dem Wohlergehen der Öffentlichkeit zu fällen; wenden Sie den Stand der Wissenschaft und Technik ihres Fachgebiets an; erbringen Sie Leistungen nur in Bereichen Ihrer eigenen Kompetenz; und beachten Sie die Prinzipien der nachhaltigen Entwicklung.

Fragen

- ❓ Ist Software Engineering auch für die Entwicklung einer nicht-komplexen Software in einem kleinen Team von zwei Personen sinnvoll?



targeted
adequate
suitable
focused

Statement: We focus on adequate and suitable solutions and approaches.

Rationale: Both solutions and approaches have to be in a reasonable proportion to the problem.

Implications: We avoid both over-engineered and cobbled-together solutions. We avoid "one-size-fits-all" approaches. We suitably adapt solutions, tools and methods.



reasoned
considered
assessed
deliberate

Statement: We think carefully and holistically in advance about our solutions and approaches.

Rationale: We always think large, even if we have to act small, because thinking in advance is more efficient and effective than correcting afterwards.

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.



up-to-date
educated
experienced
insistent

Statement: We develop high-quality solutions on the basis of up-to-date methods and technologies.

Rationale: We have to cope with the fact that the IT world is recurrently revolutionizing itself.

Implications: We continuously educate ourselves. We continuously and critically challenge and assess emerging approaches and products. We are not satisfied with mediocre solutions.



evolutionary
sustainable
harmonic
contextual

Statement: We develop sustainable solutions that optimally fit into their context.

Rationale: Nature teaches us that only evolutionary approaches and solutions have a good chance to survive in the long run.

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.

Das TRUE Manifesto gibt die Grundhaltung für gutes Software Engineering als zentrales Manifest, ähnlich dem Ansatz – aber absichtlich in teilweisem Gegensatz zum Inhalt – des populären Agilen Manifests. Das TRUE Manifesto drückt vier Hauptaspekte aus.

(T)argeted (zielgerichtet) (angemessen, passend, fokussiert): Wir fokussieren uns auf angemessene und passende Lösungen und Vorgehen, denn: Sowohl Lösungen als auch Vorgehen müssen in einem vernünftigen Verhältnis zum Problem stehen.

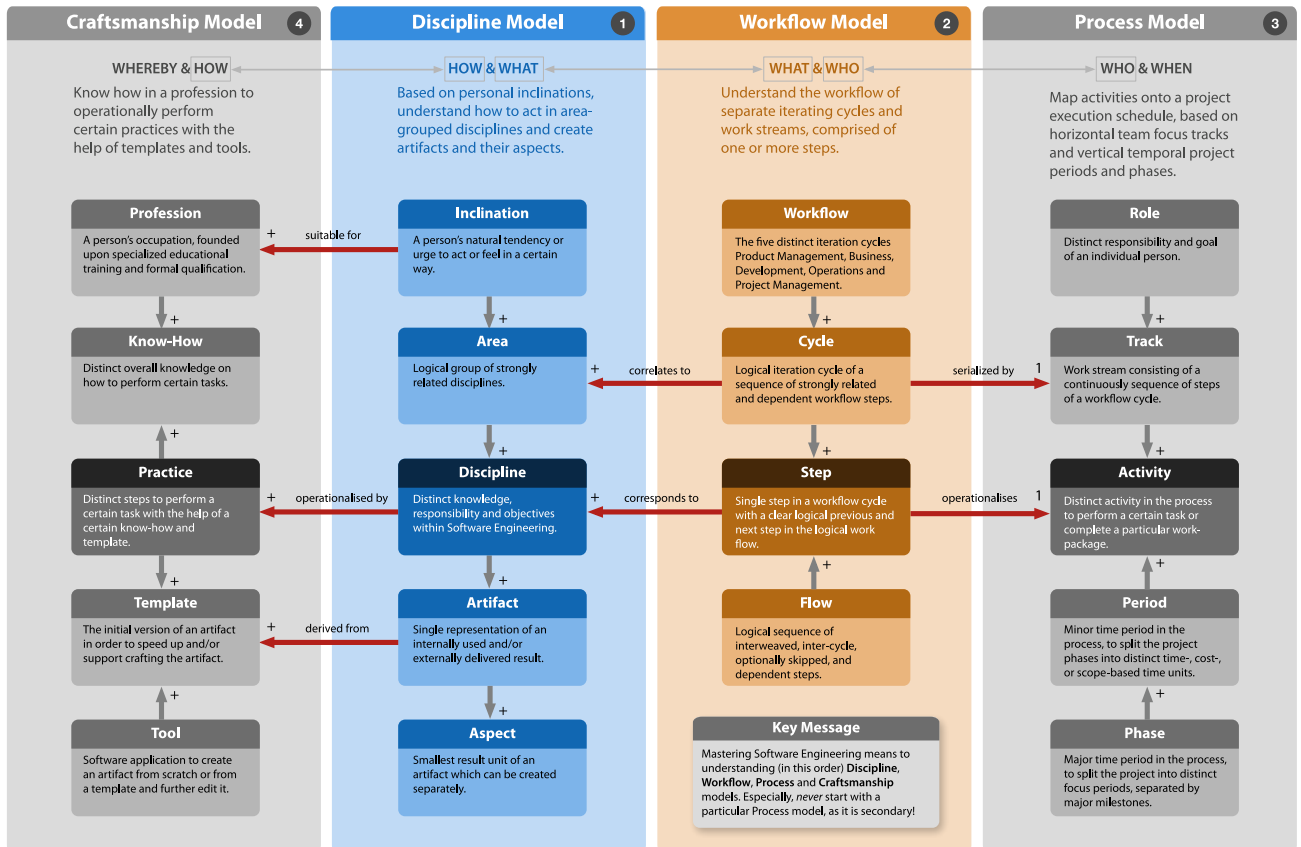
(R)easoned (durchdacht) (wohlüberlegt, bewertet, willentlich): Wir denken im Vorfeld sorgfältig über unsere geplanten Lösungen und Vorgehen nach, denn: Denken ist billiger als Korrigieren.

(U)p-to-date (zeitgemäß) (ausgebildet, erfahren, beharrlich): Wir entwickeln hochqualitative Lösungen auf der Basis zeitgemäßer Methoden und Technologien, denn: Wir müssen mit der Tatsache zurechtkommen, daß die IT-Welt sich selbst wiederkehrend umwälzt.

(E)volutionary (evolutionär) (nachhaltig, harmonisch, kontextuell): Wir entwickeln nachhaltige Lösungen, die sich optimal in ihren Kontext einfügen, denn: Die Natur lehrt uns, daß nur evolutionäre Ansätze und Lösungen eine gute Chance haben, langfristig zu überleben.

Fragen

- ? Ist ein extrem agiles Vorgehen mit ständigem Refactoring im Sinne des TRUE Manifesto für Software Engineering?



Software Engineering kann durch ein Meta-Modell verstanden werden, das auf vier verschiedenen, aber miteinander verknüpften, Modellen basiert.

Das Handwerks-Modell (**Craftsmanship Model**) ist die Basis und zielt auf das WOMIT & WIE. Es spannt den Bogen von den Berufen (**Profession**) der einzelnen Personen, deren entsprechenden **Know-Hows** und Praktiken (**Practice**) bis hin zu den zugrunde liegenden Vorlagen (**Templates**) und Werkzeugen (**Tools**).

Das Disziplinen-Modell (**Discipline Model**) zielt auf das WIE & WAS. Es trennt Software Engineering in Disziplinen (**Disciplines**), die in Bereiche (**Area**) gruppiert sind und die durch die üblichen Neigungen (**Inclination**) von einzelnen Personen motiviert sind. Jede Disziplin wird dann durch Eingabe- und Ausgabe-Artefakte (**Artifact**) und deren Aspekte (**Aspect**) beschrieben.

Das Arbeitsablauf-Modell (**Workflow Model**) zielt auf das WAS & WER. Es beschreibt einen Arbeitsablauf (**Workflow**) aus Zyklen (**Cycle**), die Schritte (**Step**) enthalten. Ein Fluss (**Flow**) ist der Durchlauf durch diese Schritte über die Zeit.

Das Prozess-Modell (**Process Model**) schließlich zielt auf das WER & WANN. Es bildet Aktivitäten (**Activity**) auf einen Projektablaufplan ab, basierend auf horizontalen Bahnen (**Track**) von Rollen (**Roles**) und vertikalen Perioden (**Period**) von Phasen (**Phase**).

Fragen

- Wieviele Zyklen kennt man im Workflow Model des Software Engineering, in denen jeweils die Personen mit ähnlichen Neigungen agieren?

ANALYSIS	AN	ARCHITECTURE	AR	CONFIGURATION	CF	ANALYTICS	AC
Software Requirements Identify Needs: We understand which outcomes of the solution are most valuable to users. <i>Requirements Engineer / Business Analyst</i>	REQ	Software Architecture Design Software: We design an orthogonal, well-balanced and well-considered solution. <i>Software Architect</i>	SWA	Software Versioning Version Artifacts: We place every artifact of the solution under strict version control. <i>Configuration Manager</i>	VER	Software Reviewing Review Code: We regularly and semantically peer-review the source code of the solution. <i>Software Tester</i>	REV
Domain Modeling Determine Solution: We model and specify the solution through involved functional and non-functional aspects. <i>Business Analyst / Business Architect</i>	DOM	System Architecture Design Systems: We ensure that the solution fits optimally into its environment. <i>System Architect / Enterprise Architect</i>	SYA	Software Assembly Assemble Artifacts: We build and package the solution through an automated and repeatable mechanism. <i>Build Manager / Build Engineer</i>	ASM	Software Testing Test Solution: We adequately test the functional and non-functional aspects of the solution. <i>Software Tester</i>	TST
business-oriented & domain-specific				constructive & technological			
EXPERIENCE User Experience Optimize Workflows: We align the solution to the perspective of the target audience. <i>User Experience Expert</i>				DEVELOPMENT Software Development Implement Code: We develop the solution outside-in, from coarse to fine aspects. <i>Software Engineer / Software Developer</i>			
EX User Interface Design User Interfaces: We design a useful, intuitive, and beautiful user interface for the solution. <i>User Interface Designer / Graphics Designer</i>				DEV Software Refactoring Refactor Code: We regularly and holistically refactor the solution to ensure long-term quality. <i>Software Engineer / Software Developer</i>			
EXPERIENCE				DEVELOPMENT			
DELIVERY Software Deployment Deploy Artifacts: We ship and deploy the solution through an automated and repeatable mechanism. <i>System Engineer</i>				COMPREHENSION Usage Documentation Document Solution: We adequately document the usage and operation of the solution. <i>Technical Writer</i>			
DL System Operations Operate Solution: We ensure that our infrastructures and the solution can be operated in a resilient and secure way. <i>System Administrator / System Operator</i>				CP User Training Train Users: We adequately train the users and operators of the solution. <i>Product Expert</i>			
DELIVERY				COMPREHENSION			
ADJUSTMENT Project Coaching Support Members: We ensure that project members use state-of-the-art methodology, technology, and tools. <i>Project Coach / Methodology Master</i>				AD Change Management Involve Stakeholders: We ensure that all stakeholders of the solution are suitably involved. <i>Change Manager</i>			
ADJUSTMENT				ADJUSTMENT			

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

[illegible]

inclinacion & knowledge

Software Engineering lässt sich durch 20 verschiedene Disziplinen (**Discipline**) (operationalisiert durch Input- und Output-Artefakte und deren Aspekte), die logisch in 10 verschiedene Bereiche (**Area**) gruppiert sind, und die wiederum logisch in 5 verschiedene Neigungen (**Inclination**) von einzelnen Personen gruppiert sind, verstehen.

Personen mit einer starken fachlichen und geschäftlichen (**domain-specific** and **business-oriented**) Neigung agieren in den Bereichen Analyse (**Analysis**) und Erfahrung (**Experience**) und in den entsprechenden Disziplinen Software-Anforderungen (**Software Requirements**), Fachmodellierung (**Domain Modeling**), Benutzererfahrung (**User Experience**) und Benutzungsschnittstelle (**User Interface**).

Personen mit einer starken konstruktiven und technologischen (**constructive** und **technological**) Neigung agieren in den Bereichen Architektur (**Architecture**) und Entwicklung (**Development**) und in den entsprechenden Disziplinen Software-Architektur (**Software Architecture**), System-Architektur (**System Architecture**), Software-Entwicklung (**Software Development**) und Software-Überarbeitung (**Software Refactoring**).

Personen mit einer starken infrastrukturellen und technologischen (**infrastructural** and **technological**) Neigung agieren in den Bereichen Konfiguration (**Configuration**) und Auslieferung (**Delivery**) und in den entsprechenden Disziplinen Software-Versionierung (**Software Versioning**), Software-Montage (**Software Assembly**), Software-Bereitstellung (**Software Deployment**) und Software-Betrieb (**Software Operations**).

Personen mit einer starken analytischen und fachlichen (**analytical** and **domain-specific**) Neigung agieren in den Bereichen Analytik (**Analytics**) und Verständnis (**Comprehension**) und in den entsprechenden Disziplinen Software-Überprüfung (**Software Reviewing**), Software-Test (**Software Testing**), Benutzungsdokumentation (**User Documentation**) und Benutzerschulung (**User Training**).

Personen mit einer starken personellen und prozessorientierten (**people-oriented** und **process-oriented**) Neigung agieren in den Bereichen Führung (**Management**) und Anpassung (**Adjustment**) und in den entsprechenden Disziplinen Projektmanagement (**Project Management**), Projektauditierung (**Project Auditing**), Projektcoaching (**Project Coaching**) und Veränderungsmanagement (**Change Management**).

Fragen

- ❓ Welche Disziplinen im Software Engineering werden als die beiden **Königdisziplinen** angesehen?