



Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall

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.



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.



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, Outlook, XMind, Firefox, Chrome, etc.



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.



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.



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.



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.



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.



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.



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.



STD

OSS

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	100%	24-48	5-25	
Software Engineering	>150	5-50	X	X	X	X	80%	>48	>25	

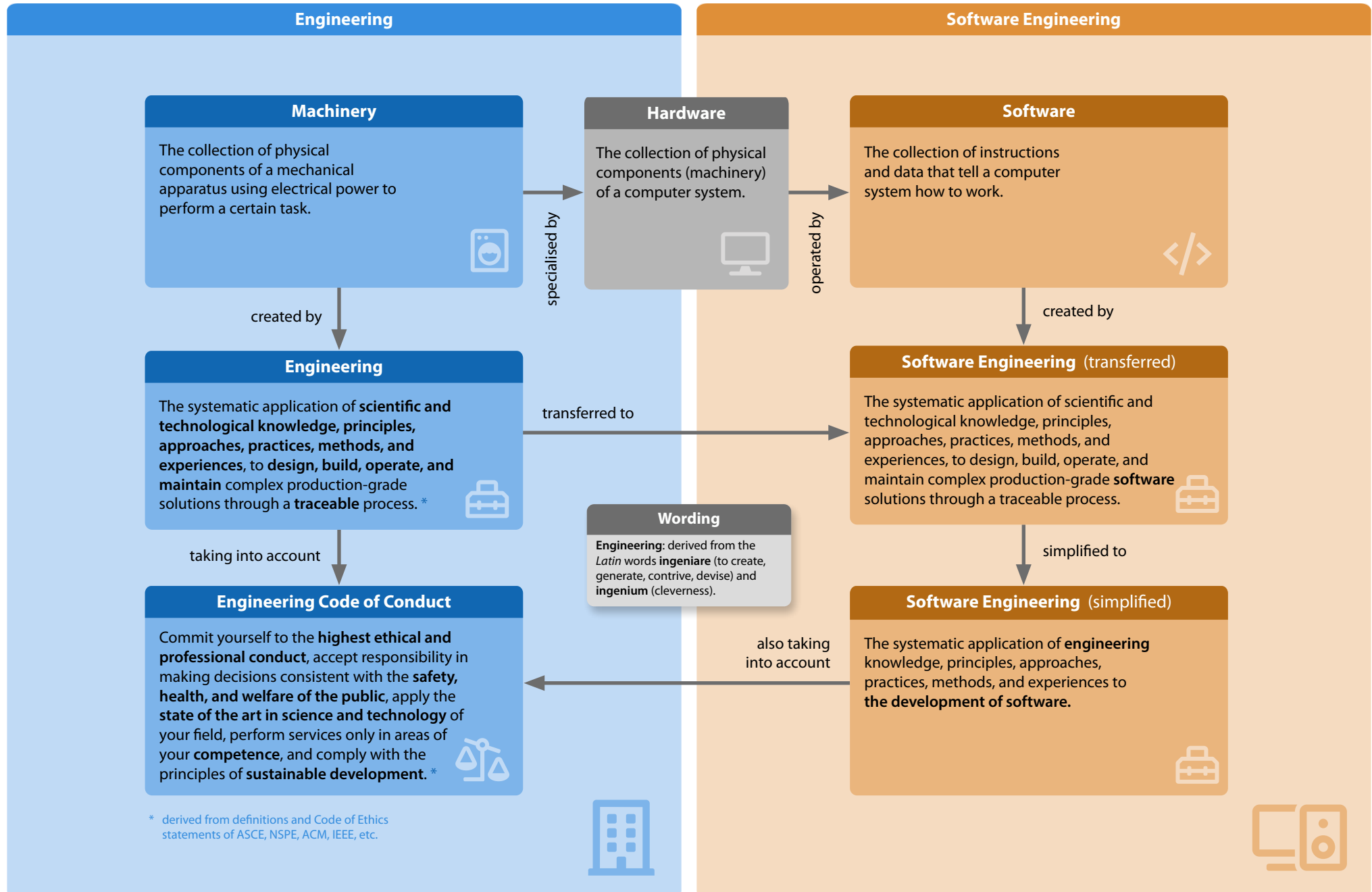
* All figures are just rough orders of magnitude for indication and illustration purposes.

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 Full 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





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.



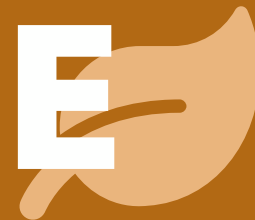
up-to-date

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.



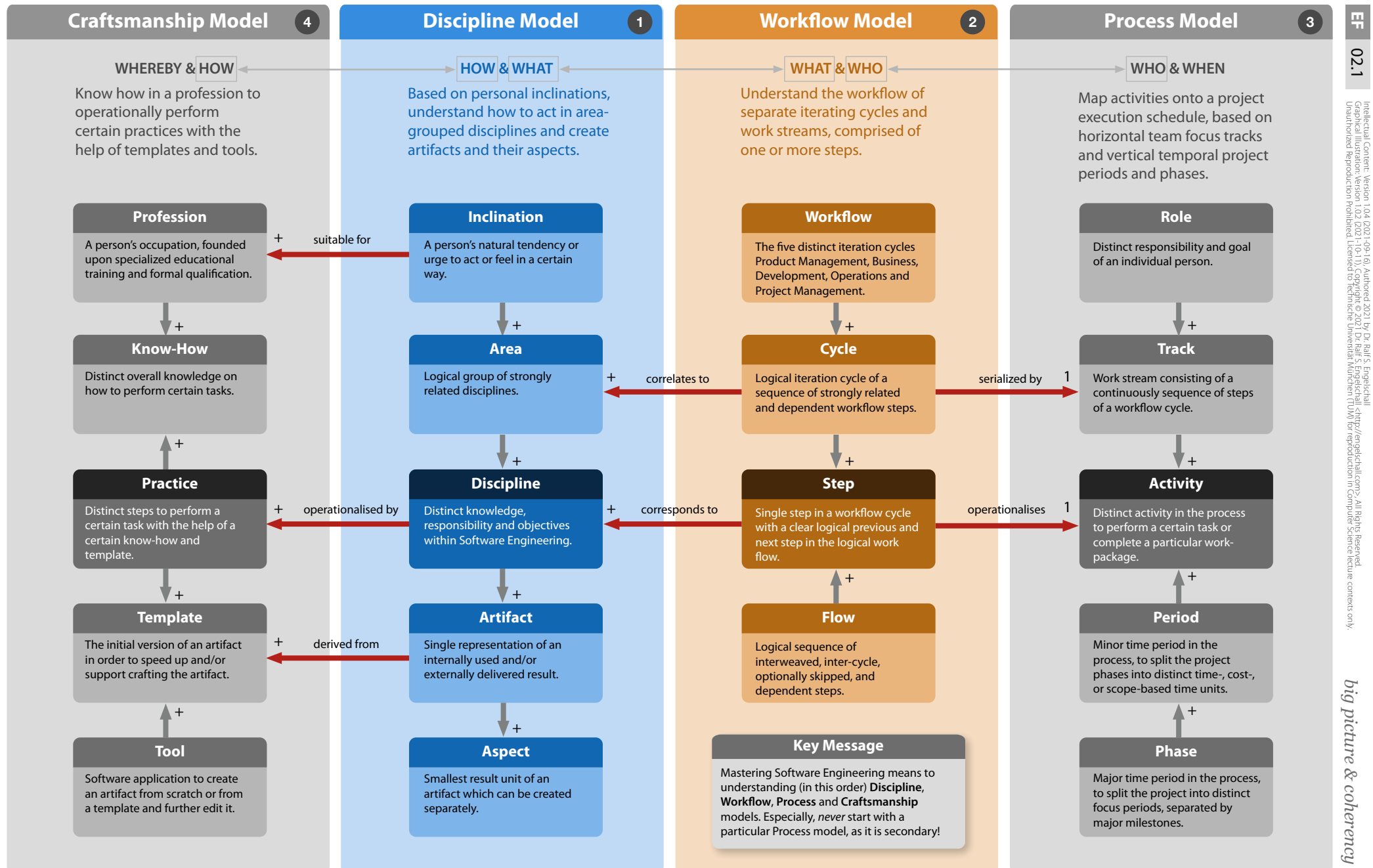
evolutionary

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.



Software Engineering Disciplines

