aspects and correlations of software architecture

Dirk Engel, info@engel-internet.de
first version 2019/01/26
last update 2022/01/26

Definitions (top left)

- **Organization**: The way in which something is done or arranged
- **System**: A regularly interacting or interdependent group of items forming a unified whole
- A **component** is functionally independent part of any system. It performs some function and may require some input or produce some output.
- **Relationship**: A continuing attachment or association between ... (not only between components but also between interfaces)
- **Environment**: The circumstances, objects, or conditions by which one is surrounded
- **Principle**: A basic idea or rule that explains or controls how something happens or works
- **Design**: The way in which something is planned and made
- **Evolution**: Mechanisms of change

Is the fundamental organization ...
- ... of a system embodied in ...
- ... its components ...
- ... their relationships to each other ...
- ... and to the environment ...
- ... and the principles guiding ...
- ... its design ...
- ... and evolution.

definition of SW architecture — aims at where? how? / aims at when?

Process (SWE.2 / SWE.5)

The purpose of the Software Architectural Design Process is to establish an architectural design and to identify which software requirements are to be allocated to which elements of the software, and to evaluate the software architectural design against defined criteria.

process purpose

a software architectural design is defined that identifies the elements of the software
- the software requirements are allocated to the elements of the software
- the interfaces of each software element are defined
- the dynamic behavior and resource consumption objectives of the software elements are defined
- consistency and bidirectional traceability are established between software requirements and software architectural design
- the software architectural design is agreed and communicated to all affected parties

process outcome — SWE.2 (ENG.5)

process definition: ISO/IEC 15504 and domain-specific models like e.g. automotiveSPICE

The purpose of the Software Integration and Integration Test Process is to integrate the software units into larger software items up to a complete integrated software consistent with the software architectural design and to ensure that the software items are tested to provide evidence for compliance of the integrated software items with the software architectural design, including the interfaces between the software units and between the software items.

process purpose — and its V-model counterpart SWE.5 (ENG.7)

aims also at **why?**

What?

Requirements

characteristics
- correct
- clear and understandable
- unambiguous
- atomic and independent
- complete and consistent
- feasible
- verifiable

key words — RFC2119: Interpretation of the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL".

types
- functional
- non-functional

functionality (FURPS)
- Capability: Size and generality of feature set
- Reusability: Compatibility, interoperability, portability
- Security: Safety and exploitability

usability
- Human factors, aesthetics, consistency, documentation, responsiveness

reliability
- Availability, predictability, robustness, stability, durability, resilience, accuracy
- Failure frequency, failure extent and time length, severity of error
- Recoverability, survivability

performance
- Speed, efficiency, resource Consumption (power, ram, cache, etc.), throughput, capacity, scalability

supportability
- Testability, flexibility (modifiability, configurability, adaptability, extensibility, modularity), installability, localizability

Quality models / attributes (criteria) — ISO-25010

- **functional suitability**: functional completeness, functional correctness, functional appropriateness
- **performance efficiency**: time behavior, resource utilization, capacity
- **compatibility**: co-existence, interoperability
- **usability**: appropriateness recognizability, learnability, operability, user error protection, user interface aesthetics, accessibility
- **reliability**: maturity, availability, fault tolerance, recoverability
- **security**: confidentiality, integrity, non-repudiation, accountability, authenticity
- **maintainability**: modularity, reusability, analyzability, modifiability, testability
- **portability**: adaptability, installability, replaceability

avoid complexity

symptoms
- change amplification: simple change requires modifications in many different places
- cognitive load: how much one need to know to complete a task
- unknown unknowns: there is something you need to know but no way to find out what it is

causes
- dependencies
- obscurity

Why?

Who?

organization

management structure
- legal department
- IT

architect
- **type**: enterprise architect, solution architect, system, software architect
- **skills**: abstract thinking, system understanding
- **people skills**: leadership, negotiation, decision making
- working with architecture teams — with and in teams

Conway's law: "organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."

stakeholder

- **developer**: design, test and productive code
- **tester**: system test, regression test
- **integrator**: Interface change management, artifacts, deliverable, integration test, continuous integration (XP), development, stabilization, branch planing, release
- **projekt management triangle**: make or buy (expertise and skills / context and experience — human resources), scope, feature set, costs, budget planning, time (quality)
- **PO, PM**
- requirements engineer
- **user** / customer

stakeholders have overlapping responsibilities

method

SAFe
- **agile team**: product a: each member the specific component she/he is working on; product b: all members the final product as one manifestation of complete product line — in a poduct line team members should always have two products in mind
- **component team** / **feature team**

XP, SCRUM, Kanban

agile vs. traditional

platform

A software platform is a set of software subsystems and interfaces that form a common structure from which a set of derivative products can be efficiently developed and produced.

Software product line engineering is a paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customisation.

product line
- patterns how to arrange platform elements (analogy: zoning regulations of cities)
- **feature model** — feature: alternatives, options, configuration, parametrization
- **use case**: 1..n per feature, scenario: one way through a use case
- **specification**: describes the details

When?

known similar systems / proof of concept (PoC) / minimal viable project (MVP)

creation with help of
- documentation
- communication

continuously
- **non-functional**: FOSS analysis (static / dynamic), KPI measurement, code analysis
- **functional**: feature set, requirements, compliance monitoring (analyze consistency), architecture and component design definition

(small) change/adaptation
- refactoring (patterns)
- optimizations
- fixes — due to review findings / because of defect
- debugging rules: 1. rule: understand the system

(big) change/adaptation — aim42: issue-costs vs. improvement-costs

evolution
- mutation (new idea)
- recombination
- selection
- genetic drift (new methodology)
- shorter cycles, higher cadence

agile vs. traditional — strategical and long-term: not (only) focussed on tomorrow but on months and even years when it comes to large projects and product line engineering

Where?

inside

- language
- **component definition**: feature responsibility, traceability, interior design (analogy: the room of a house)
- **design patterns** (interior): idioms — low-level patterns specific to a programming language

frameworks
- OS abstraction — leaky?: no blind trust, fix oversimplification when it becomes apparent
- event handling, inter thread communication (ITC), queues, timer
- watchdog
- for maintainability reasons handled like third party even if done inhouse

cross cutting

- **concepts**: naming conventions, error handling, transactions (ACID), internationalization, ...
- **(functional) safety**: e.g. ISO-26262 — QM, ASIL: Automotive Safety Integrity Level A (low) - D (high); language or compiler features (e.g. stack protection)
- **security**:
  - control flow integrity (CFI) — user with limited permissions
  - hardening by sandboxing — container: namespace restricts visibility of resources to own namespace (subset of overall resources), e.g. process and user IDs, mounted devices; cgroups limits, accounts for and isolates the resource usage; capabilities specific superuser privileges which can be independently enabled and disabled
  - fixing CVEs — Common Vulnerabilities and Exposures

(env. dependent) configuration by
- deployment: Only necessary components are deployed
- configuration management: Mapping of specific parts to client workspace
- compile time: In-line configuration via compiler switches
- **runtime**: environment variable, command line options, configuration object passed at initialization
- **public interfaces**: between interacting domains

outside / environment interaction

SW
- **power cycle management**: init / start ("boot" stages, load on demand), suspend / resume, stop / shutdown
- **persistency (CRUD)**: file system based, key/value, data base
- **logging / tracing**: Logging and tracing is not the same, both are made for different uses cases: While logging is for offline analysis of mostly critical situations, tracing is an often interactive method that gives you detailed insight about input data, internal states etc.; refer also to logging vs tracing
- middleware, inter process communication (IPC)

third party
- FOSS (take care for the license)
- dynamic linking if possible

OS
- **kernel**: monolithic, micro, L4 — GPL infected?
- user space
- real-time
- POSIX or alike — PSE51
- license costs

HW
- CPU architecture — security features
- **peripheral**: interface type, drivers, firmware

How?

decomposition

method
- middle-out: top-down, bottom-up

artifacts
- **interface**: library: narrow contract (with preconditions) over artificially wide; service: abstract also the IPC technology to support various bindings
- ISO-26262: item, system, element, composite component, component, unit, subsystem (black box, white box)

divide and conquer

SOLID
- Single-Responsibility (SRP; responsibility on function level, reason to change)
- Open-Closed (OCP): open for extension, but closed for modification
- Liskov Substitution (LSP): elements replaceable with instances of subtypes
- Interface Segregation (ISP): many client-specific interfaces are better than one general-purpose interface
- Dependency Inversion (DIP): depend upon abstractions, not details (and abstractions don't depend on details but vice versa)

OO Principles
- Separation of Concerns (SoC, concern on feature level)
- **MS: key principles**:
  - Single-Responsibility principle (SRP)
  - Principle of Least Knowledge (LoD, loose coupling)
  - Don't repeat yourself (DRY)
  - Minimize upfront design / You ain't gonna need it (YAGNI) — Keep it simple stupid (KISS)
- **others**:
  - High Cohesion
  - Design For Change
  - Information Hiding (Encapsulation) — avoid complexity with deep modules: small (general purpose) interface × lot of hidden functionality
  - Inversion of Control (IoC) — Dependency Injection (DI)

architectural patterns (analogy: the house)

follows similar rules as requirements (characteristics) and DRY

model
- **interface**: IDL, SysML
- **overall**: UML

architectural patterns
- mud-to-structure: pipe and filters, blackboard, layers, concentric architectures — IODA
- Distributed Systems: broker, client/server, master/slave, peer to peer, event bus — microservices
- Interactive Systems: model view controller (MVC), presentation abstraction control (PAC)
- Adaptable Systems: microkernel, reflection

documentation (e.g. arc42)

4+1 (semi-)formal
- **Logical**: Class Diagram, Communication/Sequence Diagram
- **Development**: Component Diagram (according to layers if applicable), Package Diagram
- **Process**: Activity Diagram
- **Physical**: Deployment Diagram — nodes, run time environment (thread priorities), artifacts, deliverables
- **Use Case**: Use Case Diagram and Scenarios

views per criteria / element
- **realization**: used language, used frameworks, libraries, used IPC binding
- **resource consumption**
- complexity, risk = severity × probability (= exposure × controllability)
- **security**: threat analysis

evaluation of alternatives: in particular not chosen alternatives and changes in rating over time

informal
- **design decisions based on**: facts and experience (evidence-based), used patterns and principles, not gut feeling and belief — loosely coupled hierarchical system of interior design, architecture, and zoning regulations to prevent from defining obstructive overall rules.

communication
- with help of documentation
- between stakeholders

is not about collecting "any numbers", to get comparable data it...
- needs a clear definition of what
- motivated by explaining the why (context and background information)
- needs a clear description of how to gain the numbers

metrics / ATAM

resource consumption
- continuous monitoring by runtime trace markers is useful because of the tragedy of the commons
- CPU vs. time, memory, IRQ, DMA

complexity
- cost efficiency
- **categories**: risk

reliability
- **impact**: if component fails the impact affects — whole system, single feature, one or more use cases, specific scenario
- **failure**: mean time between failures (MTBF), number of bugs per components
- **test coverage**: statement coverage, branch coverage, condition coverage, boundary interior coverage

tools

- modeling tool
- requirements engineering
- **lifecycle management (traceable documentation)**: work package (ticket) and planing, configuration management — continuous integration (XP), wiki
- **code generation**: interface template code from IDL — test code templates for automatic "black box" testing, component templates from overall model
- **code analysis**: static, dynamic

Aspects and correlations of (embedded) software architecture in a mind map

Dirk Engel, info@engel-internet.de – first version 2019/01/26, last update 2022/01/26

Bibliography – General

O. Vogel et al., "Software-Architektur / Grundlagen – Konzepte – Praxis", Spektrum Akademischer Verlag, 2005, ISBN 3-8274-1534-9

Mahbouba Gharbi et al., "Basiswissen für Softwarearchitekten", dpunkt.Verlag GmbH, 2018, ISBN 978-3-86490-499-8

Bibliography – Mind Map Links

4+1	https://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf (2019/01/14)
agile vs. traditional	https://agilemanifesto.org/ (2019/01/14)
aim42	https://www.aim42.org/using (2019/01/26)
arc42	https://arc42.org/ (2021/11/27)
architectural patterns	Frank Buschmann et al., "Pattern-Oriented Software Architecture / A System of Patterns", John Wiley & Sons, Ltd, 1996, ISBN 978-0-471-95869-7
ATAM	https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5177 (2021/12/18)
avoid complexity	John Ousterhout, "A Philosophy of Software Design", Yaknyam, 2018, ISBN 978-1-7321022-0-0
capabilities	http://man7.org/linux/man-pages/man7/capabilities.7.html (2019/01/14)
categories	Johannes Siedersleben, "Moderne Softwarearchitektur", dpunkt.verlag GmbH, 2004, ISBN 3-89864-292-5
CFI	https://source.android.com/devices/tech/debug/cfi (2019/01/14)
cgroups	https://en.wikipedia.org/wiki/Cgroups (2019/01/14)
characteristics	http://www.informit.com/articles/article.aspx?p=1152528&seqNum=4 (2019/01/14)
component	https://www.techopedia.com/definition/3217/component (2019/01/07)
concentric architectures	https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (2020/12/26)
container	https://jvns.ca/blog/2016/10/10/what-even-is-a-container/ (2019/01/15)
Conway's law	https://en.wikipedia.org/wiki/Conway%27s_law (2019/01/14)
CVEs	https://cve.mitre.org/ (2019/01/04)
debugging rules	David J. Agans, "Debugging", AMACOM, 2002, ISBN 0-8144-7168-4
design	https://dictionary.cambridge.org/de/worterbuch/englisch/design (2019/01/07)
design patterns	Erich Gamma et al., "Design Patterns / Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994, ISBN 0-201-63361-2
enterprise architect	Wolfgang Goebl et al., "Enterprise Design Patterns / 35 Ways to Radically Increase Your Impact on the Enterprise", Intersection Group, 2020.
environment	https://www.merriam-webster.com/dictionary/environment (2019/01/07)
evidence-based	http://www.knosof.co.uk/ESEUR/ESEUR-mobile.pdf (version 1.0 from 2020/11/08)
evolution	http://nectunt.bifi.es/to-learn-more-overview/mechanisms-of-evolutionary-change/ (2019/01/14)
FURPS	https://en.wikipedia.org/wiki/FURPS (2019/01/14)
GPL infected	https://yarchive.net/comp/linux/gpl_modules.html (2019/01/14)
human resources	https://github.com/dirkengel/articles/blob/main/FTEbasedPlanning.pdf (2018/10/11)
idioms	https://www.oreilly.com/library/view/pattern-oriented-software-architecture/9781118725269/OEBPS/9781118725269_c04.htm (2019/01/14)
Information Hiding	https://en.wikipedia.org/wiki/Information_hiding (2019/01/14)
IODA	https://leanpub.com/ioda-architektur-im-vergleich-dnp (2020/12/15)
ISO-25010	https://iso25000.com/index.php/en/iso-25000-standards/iso-25010 (2019/01/14)
ISO-26262	https://www.iso.org/standard/43464.html (2019/01/14)
organization	https://dictionary.cambridge.org/de/worterbuch/englisch/organization (2019/01/07)
obstructive	Stefan Toth, "Wie Architektur agile Zusammenarbeit fördert oder behindert", OBJEKTspektrum 01/2018
patterns	Christopher Alexander et al., "A Pattern Language: Towns, Buildings, Construction", Oxford University Press, 1977, ISBN 0-19-501919-9
people skills	https://www.oreilly.com/ideas/4-essential-skills-software-architects-need-to-have-but-often-dont (2019/01/04)
platform	Klaus Pohl et al.,"Software Product Line Engineering", Springer, 2005, ISBN 978-3540243724
principle	https://dictionary.cambridge.org/dictionary/english/principle (2019/01/07)
process definition	https://en.wikipedia.org/wiki/ISO/IEC_15504 (2019/01/14)
key principles	https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658124(v=pandp.10)#KeyDesignPrinciples (2021/11/12)
leaky	https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/ (2021/11/12)
logging vs. tracing	https://en.wikipedia.org/wiki/Tracing_(software)#Event_logging_versus_tracing (2019/01/08)
management triangle	https://github.com/dirkengel/articles/blob/main/QualityAndTheProjectManagementTriangle.pdf (2021/10/24)
metrics (general)	Kan S. H. "Metrics and Models in Software Quality Engineering", Addison-Wesley, 2nd ed. 2002.
	https://onlinelibrary.wiley.com/doi/epdf/10.1002/smr.2296 (2022/01/26)
metrics (tracking)	https://github.com/dirkengel/articles/blob/main/TheOzoneHoleMetric.pdf (2021/10/24)
	https://www.thoughtworks.com/insights/articles/fitness-function-driven-development (2021/10/24)
namespace	http://man7.org/linux/man-pages/man7/namespaces.7.html (2019/01/15)
narrow contract	http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p1743r0.pdf (2021/12/03)
microservices	Mark Richards, "Software Architecture Patterns", O'Reilly Media, Inc., 2015, ISBN 9781491924242
refactoring (patterns)	Martin Fowler, "Refactoring", Addison-Wesley, 1999, ISBN 0-201-48567-2
relationship	https://www.collinsdictionary.com/de/worterbuch/englisch/relationship (2019/01/07)
RFC2119	https://www.ietf.org/rfc/rfc2119.txt (2019/01/14)
SOLID	https://en.wikipedia.org/wiki/SOLID (2019/01/14)
SW architecture	http://www.iso-architecture.org/ieee-1471/defining-architecture.html (2019/01/14)
SWE.2	https://flecsim.de/images/download/AutomotiveSpice/Automotive%20Spice%203.0/SWE.2.html (2019/01/14)
SWE.5	https://flecsim.de/images/download/AutomotiveSpice/Automotive%20Spice%203.0/SWE.5.html (2019/01/14)
system	https://www.merriam-webster.com/dictionary/system (2019/01/07)
team	Thorsten Janning, "Scaling Agility: Wie führt man 100 agile Teams?" OBJEKTspektrum, Ausgabe Agility/2014
test coverage	Peter Liggesmeyer, "Software-Qualität", Spektrum Akad. Verlag 2002, ISBN 3-8274-1118-1
tragedy of the commons	https://en.wikipedia.org/wiki/Tragedy_of_the_commons (2020/12/26)
unknown unknowns	The Five Orders of Ignorance: http://www.corvusintl.com/CACM002-5OI.htm (2021/10/24)
XP	Kent Beck, "Extreme Programming Explained", Addison-Wesley, 2000, ISBN 0-201-61641-6