aspects and correlations of software architecture

Dirk Engel, info@engel-internet.de
first version 2019/01/26
last update 2022/10/27

What?

definition of SW architecture

aims at where? how?

- ***Concept***: An abstract or generic idea generalized from particular instances. — **Fundamental concepts or ...**
- ***System***: A regularly interacting or interdependent group of items forming a unified whole. — **... properties of a system ...**
- Environment : The circumstances, objects, or conditions by which one is surrounded. — **... in its environment ...**
- ***Element***: a basic member of a mathematical or logical class or set. — **... embodied in its elements, ...**
- Relationship : A continuing attachment or association (***interface***) between elements. — **... relationships, ...**
- ***Principle***: A basic idea or rule that explains or controls how something happens or works. — **... and in the principles of ...**
- Design : The way in which something is planned and made. — **... its design ...**
- Evolution : Mechanisms of ***change***. — **... and evolution.**

aims at when?

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

aims at why?

- process purpose
  - 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 outcome — SWE.2
  - 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 purpose — and its V-model counterpart SWE.5
  - 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.

Why?

- ***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 (criteria)
- quality model / ***criteria*** — FURPS
  - functionality
    - Capability: Size and generality of feature set
    - Reusability: Compatibility, interoperability, portability
    - Security : Safety and exploitability
  - usability
    - Human factors, aesthetics, consistency, documentation, responsiveness
  - ***reliability***
    - Availability, predicatbility, robustness, stability, durability, resilience, accuracy
    - Failure frequency, failure extent and time length, serverity of error
    - Recoverability, survivability
  - performance
    - Speed, efficiency, ***resource consumption*** (power, ram, cache, etc.), throughput, capacity, scalability
  - supportabiliy
    - Testability, flexibility (modifiability, configurability, adaptability, extensibility, modularity), installability, localizability
- quality model / ***criteria*** — ISO-25010
  - functional suitability
    - functional completeness, functional correctness, functional appropriateness
  - performance efficiency
    - time behavior, resource utilization (resource consumption), 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

Who?

- organization
  - management
  - structure
    - Conway's law : "organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."
  - legal department
  - IT
- stakeholder
  - architect
    - type
      - enterprise architect , solution architect
      - system, software architect
    - skills
      - abstract thinking
      - system understanding
      - people skills
        - with and within teams
        - team-minded
        - decision making
        - leadership
        - negotiation
  - developer
    - design
    - test and productive code
  - integrator
    - interface change management
    - artifacts , ***deliverable***
    - ***continuous integration*** (XP)
    - integration test of components and (sub-) systems
    - branch planing
      - (big) change
      - stabilization
      - release
  - tester
    - regression test
    - system test
  - PO, PM
    - projekt management triangle
      - scope, feature set
      - backlog prioritization
        - WSJF
        - technical risk
        - reduction
        - ***business value***
      - time
      - (quality)
      - ***costs*** , budget planning
        - make or buy (third party)
        - human resources
          - expertise and skills
          - context and experience
  - requirements engineer
  - user
  - customer
- agile vs. traditional
  - stakeholders have overlapping responsibilities (DevOps infinity loop)
  - agile team
    - component team
    - feature team
  - SAFe
  - method
    - XP, SCRUM, Kanban
  - In a product line team, members must always balance the special requirements of individual product instances with the product line's variability goals (at component level).

When?

- product line
  - platform
    - A software platform is a set of software systems 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.
  - patterns how to arange platform elements (analogy: zoning regulations of cities)
  - feature model
    - feature
    - configuration
      - alternatives
      - options
      - parametrization
    - use case
      - 1..n per feature
      - scenario
        - one way through a use case
      - specification
        - describes the details
  - creation with help of
    - known similar systems
    - proof of concept (PoC)
    - minimal viable project (MVP)
- continuously (via tools)
  - documentation
  - communication
  - compliance monitoring (analyze consistency)
    - requirements
      - code analysis
        - static
        - FOSS analysis (license)
        - dynamic
        - KPI measurement
      - non-functional
      - functional
        - feature set
      - design definition
        - architecture and component
  - change /adaptation
    - due to review findings
      - optimizations
        - refactoring (patterns)
      - fixes
    - because of defect
      - 1. rule: system understanding
        - debugging rules
- (big) change /adaptation
  - issue- costs vs. improvement- costs (aim42)
  - evolution
    - mutation (new idea)
    - recombination
    - selection
    - genetic drift (new methodology)
- agile vs. traditional
  - shorter cycles, higher cadence
- strategical and long-term
  - not focussed on tomorrow but on months and even years when it comes to large projects and product line engineering (costs)

Where?

- inside
  - component definition
    - feature responsibility — traceability
    - interior design (analogy: the room of a house)
      - design patterns
      - idioms — programming language specific low-level patterns
  - ***programming language*** — control, performance, safety, scalability vs. community, experience, experise
  - frameworks (to DRY)
    - OS abstraction — leaky ? no blind trust, fix oversimplification when it becomes apparent
    - event handling, inter thread communication (ITC), queues, timer
    - SW watchdog (threads)
    - for maintainability reasons handled like third party even if done inhouse
- cross cutting criteria
  - concepts — naming conventions, error/exception handling, transactions (ACID), internationalization, ...
  - (functional) safety — ISO-26262 — QM, ASIL: Automotive Safety Integrity Level A (low) - D (high)
  - security
    - cybersecurity engineering — ISO-21343
    - programming language or compiler features e.g. stack protection, control flow integrity
    - hardening by sandboxing
      - user with limited permissions
      - 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 consumption
      - capabilities — specific superuser privileges which can be independently enabled and disabled
    - fixing CVEs — Common Vulnerabilities and Exposures
- outside/ environment interaction
  - HW
    - CPU architecture — security features
    - peripheral
      - interface type — drivers
      - firmware
  - SW
    - ***OS***
      - POSIX or alike — PSE51
      - kernel
        - monolithic, micro, L4
        - real-time — challenges using Linux
        - GPL infected?
      - user space
      - license constraints and costs
    - ***third party***
      - FOSS — take care for the ***license***
      - dynamic linking if possible (maintainability)
    - infrastructure
      - power cycle management
        - init / start — "boot" stages, load on demand
        - suspend / resume
        - stop / shutdown
      - persistency (CRUD)
        - file system based
        - key/value, data base
      - middleware, inter process communication (IPC)
      - logging/ tracing — Both serve different uses cases: Logging records critical situations for offline analysis, tracing is an (interactive) method to gain insight into internals; see also logging vs tracing
    - (env. dependent) ***configuration*** by
      - deployment: Only necessary components are deployed
      - workspace mapping of specific parts via configuration management
      - compile time: In-line configuration via compiler switches
      - runtime
        - environment variable
        - command line options
        - configuration object passed at initialization
    - public interfaces — between interacting domains

How?

- divide and conquer
  - decomposition
    - method
      - middle-out
      - top-dowr
      - bottom-up
    - ***artifacts*** — ISO-26262
      - element — system or HW/SW component or HW part or SW unit
      - item — system or combination of (sub-) systems — (composite) ***component*** — unit / subcomponent
      - interface
        - library: narrow contract (with defense programming) over artificially wide
        - service: abstract also the IPC technology to support various bindings (IDL)
  - OO principles
    - MS: key principles
      - Separation of Concerns (SoC, concern on feature level)
      - 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)
    - SOLID
      - Single-Responsibility (SRP, responsibiliy 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 Segragation (ISP): many client-specific interfaces are better than one general-purpose interface
      - Dependency Inversion (DIP): High level modules don't depend on low level, both depend on abstractions
      - vs. CUPID properties — Composable, Unix philosophy (one thing well), Predictable, Idiomatic, Domain-based
    - 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)
- ***documentation*** and doing (walk the talk)
  - follows the DRY rule and the requirement characteristics to explain the architecture by ...
    - definition: priorities of goals (ATAM) and applied principles , concepts , patterns
    - manifestation: illustrated with help of perspectives and views
    - implementation: proven with help of perspectives and views revealing the target/actual comparison
  - frameworks — arc42, SEI views and beyond
  - semi-formal
    - architectural patterns (analogy: the house)
      - mud-to-structure — pipe and filters, blackboard, layers, concentric architectures
      - 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
    - tactics-based questionnaires
    - design decisions
      - based on facts and experience (evidence-based , metrics) *not* gut feeling and belief
      - prioritization between quality attributes/ criteria (ATAM and its Utility Tree)
        - ***technical risk***
        - business value
      - evaluated alternatives — documents not chosen alternatives and changes in rating over time
      - used ***patterns*** and principles (pattern language) — loosely coupled hierarchical system of interior design, architecture, zoning regulations (analogies) to avoid obstructive overall rules.
  - formal
    - ***model***
      - interface — IDL
      - general — SysML, UML
    - views
      - types 4+1
        - logical (functionality) — class, communication/sequence diagram
        - development
          - package diagram — artifacts , deliverables
          - component diagram
            - resource consumption
            - security — threat analysis
            - safety: risk — correlates with complexity
            - ***risk*** = severity × probability (= exposure × likelihood)
        - process (dynamics) — activity diagram
        - physical — deployment diagram — run time evironment, nodes, processes, threads, IPC binding
        - use case — cse case diagram and scenarios
      - levels C4
        - context, system boundaries and relationships with users
        - containers as representations of applications or a data stores
        - components , decompose containers into interrelated components
        - code, architectural elements mapped to code
      - concerns — different types at different levels with details on different concerns
- communication
  - with help of documentation
  - between stakeholders
- (criteria) metrics via tools
  - to get comparable data the metrics need ... (and be careful in seeing correlations)
    - ... to be motivated by explaining the why (context and background information).
    - ... a clear definition on how to achieve the desired numbers.
    - ... a clear description about the tool, how to use, and which data to record.
  - resource consumption
    - continuous monitoring because of the tragedy of the commons
    - CPU vs. time, memory, IRQ, DMA
  - complexity
    - cost efficiency
    - categories — risk
  - reliability
    - failure
      - impact if component fails — whole system, single feature, one or more use cases, specific scenario
      - mean time between failures (MTBF)
      - number of bugs per components
    - test coverage
      - statement coverage
      - branch coverage
      - condition/decision coverage
      - boundary interior coverage
- ***tools***
  - lifecycle management (traceable documentation)
    - requirements engineering
    - work package (ticket) and planing
    - model ing
    - configuration management
      - docs as code
      - 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 2022/10/26

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
C4	https://c4model.com/ and https://en.wikipedia.org/wiki/C4_model (2022/07/31)
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
cgroups	https://en.wikipedia.org/wiki/Cgroups (2019/01/14)
challenges	“Challenges Using Linux as a Real-Time Operating System”; https://ntrs.nasa.gov/citations/20200002390 (2022/03/19)
characteristics	http://www.informit.com/articles/article.aspx?p=1152528&seqNum=4 (2019/01/14)
concentric architectures	https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (2020/12/26)
concept	https://www.merriam-webster.com/dictionary/concept (2022/05/14)
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)
CUPID	"SOLID vs. CUPID”: https://youtu.be/2QahGarHpXQ (2022/02/05)
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 decisions	“A documentation framework for architecture decisions”; http://www.iso-architecture.org/viewpoints/docs/Heesch_Avgeriou_Hilliard-JSS-2011b.pdf (2022/05/14)
design patterns	Erich Gamma et al., “Design Patterns / Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1994, ISBN 0-201-63361-2
docs as code	https://www.writethedocs.org/guide/docs-as-code/ (2022/10/22)
element	https://www.merriam-webster.com/dictionary/element (2022/05/14)
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)
hierachical	“The C4 model for visualising software architecture: Context, Containers, Components, and Code”; https://c4model.com/ (2022/05/14)
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)
ISO-21343	https://www.iso.org/standard/70918.html (2022/10/26)
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)
key principles	https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658124(v=pandp.10)#KeyDesignPrinciples (2021/11/12)
language (pattern)	Christopher Alexander et al., "A Pattern Language: Towns, Buildings, Construction", Oxford University Press, 1977, ISBN 0-19-501919-9
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.
	“Metrics in automotive software development: A systematic literature review”; 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)
microservices	Mark Richards, “Software Architecture Patterns”, O'Reilly Media, Inc., 2015, ISBN 9781491924242
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)
obstructive	Stefan Toth, “Wie Architektur agile Zusammenarbeit fördert oder behindert”, OBJEKTspektrum 01/2018
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)
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)
SEI views	https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484159 (2022/05/14)
SOLID	https://en.wikipedia.org/wiki/SOLID (2019/01/14)
SW architecture	https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:42010:ed-1:v1:en, section 3.2, (2022/05/14)
SWE.2	https://www.flecsim.de/images/download/AutomotiveSpiceShortened/Automotive%20Spice%203.1/SWE.2.html (2022/10/26)
SWE.5	https://www.flecsim.de/images/download/AutomotiveSpiceShortened/Automotive%20Spice%203.1/SWE.5.html (2022/10/26)
system	https://www.merriam-webster.com/dictionary/system (2019/01/07)
tactics	Len Bass et al. “Software Architecture in Practice”, Addison-Wesley, 4th ed., 2021.
team	Thorsten Janning, “Scaling Agility: Wie führt man 100 agile Teams?” OBJEKTspektrum, Ausgabe Agility/2014
test coverage	Peter Liggesmeyer, “Sofware-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	https://www.researchgate.net/publication/27293624_The_Five_Orders_of_Ignorance (2022/10/26)
why	Robert Cialdini, “Influence, New and Expanded: The Psychology of Persuasion”, 2021.
WSJF	https://www.scaledagileframework.com/wsjf/ (2022/02/12)
XP	Kent Beck, “Extreme Programming Explained”, Addison-Wesley, 2000, ISBN 0-201-61641-6