aspects and correlations of software architecture

Dirk Engel, info@engel-internet.de
first version 2019/01/26
last update 2021/01/16

What? — definition of SW architecture

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

aims at where? how?
aims at when?

Why?

process definition — aims also at why?
ISO/IEC 15504 and domain-specific models like e.g. automotiveSPICE

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)

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)

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

- 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, 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
- ISO-25010
  - functional suitability — functional completeness, functional correctness, functional appropriateness
  - reliability — maturity, availability, fault tolerance, recoverability
  - usability — appropriateness, recognizability, learnability, operability, user error protection, user interface aesthetics, accessibility
  - performance efficiency — time behavior, resource utilization, capacity
  - security — confidentiality, integrity, non-repudiation, accountability, authenticity
  - maintainability — modularity, reusability, analyzability, modifiability, testability
  - compatibility — co-existence, interoperability
  - portability — adaptability, installability, replaceability

Who? — stakeholder

architect
- soft skills
  - system understanding
  - abstract thinking
- people skills
  - leadership
  - negotiation
  - decision making
  - working with architecture teams

organization
- Conway's law: "organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organizations."
- management
- structure
- legal department
- IT

customer
- user
- requirements engineer

PO, PM
- budget planning
- projekt management triangle
  - scope, feature set
  - costs
    - make or buy
    - human resources — expertise and skills; context and experience
  - time (quality)

integrator
- artifacts, deliverable
- continuous integration (XP)
- integration test

developer
- development
  - branch planing
  - stabilization
  - release
- productive code
  - unit test

tester
- system test
- regression test

agile vs. traditional
- stakeholders have overlapping responsibilities
- SAFe — in a poduct line team members should always have two products in mind
  - 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
  - agile team
    - component team
    - feature team
  - method — XP, SCRUM, Kanban

Where?

inside
- component definition
  - language
  - feature responsibility
  - interior design (analogy: the room of a house) — (interior) design patterns; idioms — low-level patterns specific to a programming language
- frameworks
  - OS abstraction
  - event handling, inter thread communication (ITC), queues, timer
  - watchdog
  - for maintainability reasons handled like third party even if done inhouse

cross cutting
- error handling
- transactions
- (internationlization)
- security
  - language or compiler features (e.g. stack protection)
  - control flow integrity (CFI)
  - user with limited permissions
  - sand boxing
    - 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
- (functional) safety — e.g. ISO-26262 — ASIL: Automotive Safety Integrity Level A (low) - D (high)
- quality

outside / environment interaction

- (env. dependent) configuration by
  - deployment: Only necessary components are deployed
  - configuration management: Project specific parts are mapped to client workspace
  - compile time: In-line configuration via compiler switches
  - environment variable
  - runtime
    - command line options
    - configuration object passed at initialization
- SW
  - public interfaces — between interacting domains
  - infrastructure
    - power cycle management
      - "boot" stages — init / start; load on demand
      - suspend / resume
      - stop / shutdown
    - persistency — file system based; key/value, data base
    - logging / tracing — Logging and tracing is not the same, both are made for different use 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
    - kernel space — GPL infected?
    - user space
  - OS
    - real-time
    - POSIX or alike
    - license costs
- HW
  - CPU architecture — security features
  - peripheral — interface type — drivers; firmware

How?

decomposition
- method
  - top-down
  - middle-out
  - bottom-up
- interface — abstract not only functionality (i.e. provided requests/responses/attributes) but also abstract the IPC technology in oder to support various bindings
- artifacts — 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 Segragation (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
  - 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)
  - others
    - High Coherence
    - Design for Change
    - Information Hiding (Encapsulation)
    - Inversion of Control (IoC) — Dependency Injection (DI)

documentation
- follows similar rules as requirements (characteristics) and DRY
- model
  - interface — IDL
  - overall — SysML; UML
- architectural patterns (analogy: the house)
  - mud-to-structure — pipe and filters, blackboard, layers, concentric architectures , and 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
- formal (language)
  - 4+1
    - 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 evironment — thread priorities; artifacts, deliverables
    - Use Case — Use Case Diagram and Scenarios
  - views
    - realization — used language; used frameworks, libraries; used IPC binding
    - security — threat analysis
    - resource consumption — complexity, risk, and error impact — component based
    - evaluation of alternatives
- informal
  - design decisions based on — facts and experience — 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

metrics
- is not about collecting "any numbers", to get comparable data it…
  - needs a clear definition of what
  - should be motivated by explaining the why (context and background information)
  - needs a clear description of how to gain the numbers
- 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
- 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
- reliability
- test coverage
  - statement coverage
  - branch coverage
  - condition coverage
  - boundary interior coverage

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

When?

continuously
- creation with help of
  - known similar systems
  - proof of concept (PoC)
  - minimal vital project (MVP)
- product line
  - 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.
    - patterns how to arrange platform elements (analogy: zoning regulations of cities)
  - feature model
    - configuration
      - alternatives
      - options
      - parametrization
    - feature
    - use case — 1..n per feature
    - specification
      - scenario — one way through a use case
      - describes the details
- requirements
  - documentation
  - communication
  - non-functional
    - code analysis — FOSS analysis (static); KPI measurement (dynamic)
    - compliance monitoring (analyze consistency)
  - functional — feature set
- architecture and component design definition

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

(big) change/adaptation
- issue-costs vs. improvement-costs (aim42)
- evolution
  - Mutation (new idea)
  - Recombination
  - Selection
  - Gen drift (new methodology)
- agile vs. traditional
  - shorter cycles, higher cadence
  - 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

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

Dirk Engel, info@engel-internet.de – first version 2019/01/26, last update 2020/12/26

Bibliography – General

O. Vogel, I. Arnold, A. Chughtai, E. Ihler, U. Mehlig, Th. Neumann, M. Völter, U. Zdun, "Software-Architektur / Grundlagen – Konzepte – Praxis", Spektrum Akademischer Verlag, 2005, ISBN 3-8274-1534-9

Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke, "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)
architectural patterns	Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, "Pattern-Oriented Software Architecture / A System of Patterns", John Wiley & Sons, Ltd, 1996, ISBN 978-0-471-95869-7
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, John Vlissides, Ralph Johnson und Richard Helm, "Design Patterns / Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994, ISBN 0-201-63361-2
environment	https://www.merriam-webster.com/dictionary/environment (2019/01/07)
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://www.linkedin.com/pulse/why-fte-based-planning-useful-managing-sw-engineering-dirk-engel/
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, Sara Ishikawa, Murray Silverstein, "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
logging vs. tracing	https://en.wikipedia.org/wiki/Tracing_(software)#Event_logging_versus_tracing (2019/01/08)
namespace	http://man7.org/linux/man-pages/man7/namespaces.7.html (2019/01/15)
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, "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)
XP	Kent Beck, "Extreme Programming Explained", Addison-Wesley, 2000, ISBN 0-201-61641-6