WIKIPEDIA

Software architecture

Software architecture refers to the high level structures of a software system and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations.[1] The *architecture* of a software system is a metaphor, analogous to the architecture of a building.[2] It functions as a blueprint for the system and the developing project, laying out the tasks necessary to be executed by the design teams.[3]

Software architecture is about making fundamental structural choices which are costly to change once implemented. Software architecture choices include specific structural options from possibilities in the design of software. For example, the systems that controlled the space shuttle launch vehicle had the requirement of being very fast and very reliable. Therefore, an appropriate real-time computing language would need to be chosen. Additionally, to satisfy the need for reliability the choice could be made to have multiple redundant and independently produced copies of the program, and to run these copies on independent hardware while cross-checking results.

Documenting software architecture facilitates communication between stakeholders, captures early decisions about the high-level design, and allows reuse of design components between projects.[4]:29–35

Contents

Scope

Opinions vary as to the scope of software architectures:[5]

- *Overall, macroscopic system structure*;[6] this refers to architecture as a higher level abstraction of a software system that consists of a collection of computational *components* together with *connectors* that describe the interaction between these components.
- *The important stuff—whatever that is*;[7] this refers to the fact that software architects should concern themselves with those decisions that have high impact on the system and its stakeholders.
- *That which is fundamental to understanding a system in its environment*"[8]
- *Things that people perceive as hard to change*;[7] since designing the architecture takes place at the beginning of a software system's lifecycle, the architect should focus on decisions that "have to" be right the first time. Following this line of thought, architectural design issues may become non-architectural once their irreversibility can be overcome.
- *A set of architectural design decisions*;[9] software architecture should not be considered merely a set of models or structures, but should include the decisions that lead to these particular structures, and the rationale behind them. This insight has led to substantial research into software architecture knowledge management.[10]

There is no sharp distinction between software architecture versus design and requirements engineering (see Related fields below). They are all part of a "chain of intentionality" from high-level intentions to low-level details.[11]:18

Characteristics

Software architecture exhibits the following:

Multitude of stakeholders: software systems have to cater to a variety of stakeholders such as business managers, owners, users, and operators. These stakeholders all have their own concerns with respect to the system. Balancing these concerns and demonstrating how they are addressed is part of designing the system.[4]:29–31 This implies that architecture involves dealing with a broad variety of concerns and stakeholders, and has a multidisciplinary nature.

Separation of concerns: the established way for architects to reduce complexity is to separate the concerns that drive the design. Architecture documentation shows that all stakeholder concerns are addressed by modeling and describing the architecture from separate points of view associated with the various stakeholder concerns.[12] These separate descriptions are called architectural views (see for example the 4+1 Architectural View Model).

Quality-driven: classic software design approaches (e.g. Jackson Structured Programming) were driven by required functionality and the flow of data through the system, but the current insight[4]:26–28 is that the architecture of a software system is more closely related to its quality attributes such as fault-tolerance, backward compatibility, extensibility, reliability, maintainability, availability, security, usability, and other such –ilities. Stakeholder concerns often translate into requirements on these quality attributes, which are variously called non-functional requirements, extra-functional requirements, behavioral requirements, or quality attribute requirements.

Recurring styles: like building architecture, the software architecture discipline has developed standard ways to address recurring concerns. These "standard ways" are called by various names at various levels of abstraction. Common terms for recurring solutions are architectural style,[11]:273–277 tactic,[4]:70–72 reference architecture[13][14] and architectural pattern.[4]:203–205

Conceptual integrity: a term introduced by Fred Brooks in *The Mythical Man-Month* to denote the idea that the architecture of a software system represents an overall vision of what it should do and how it should do it. This vision should be separated from its implementation. The architect assumes the role of "keeper of the vision", making sure that additions to the system are in line with the architecture, hence preserving conceptual integrity.[15]:41–50

Cognitive constraints: an observation first made in a 1967 paper by computer programmer Melvin Conway that organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations. As with conceptual integrity, it was Fred Brooks who introduced it to a wider audience when he cited the paper and the idea in his elegant classic *The Mythical Man-Month*, calling it "Conway's Law."

Motivation

Software architecture is an "intellectually graspable" abstraction of a complex system.[4]:5–6 This abstraction provides a number of benefits:

- *It gives a basis for analysis of software systems' behavior before the system has been built.*[2] The ability to verify that a future software system fulfills its stakeholders' needs without actually having to build it represents substantial cost-saving and risk-mitigation.[16] A number of techniques have been developed to perform such analyses, such as ATAM.
- *It provides a basis for re-use of elements and decisions.*[2][4]:35 A complete software architecture or parts of it, like individual architectural strategies and decisions, can be re-used across multiple systems whose stakeholders require similar quality attributes or functionality, saving design costs and mitigating the risk of design mistakes.
- *It supports early design decisions that impact a system's development, deployment, and maintenance life.*[4]:31 Getting the early, high-impact decisions right is important to prevent schedule and budget overruns.
- *It facilitates communication with stakeholders, contributing to a system that better fulfills their needs.*[4]:29–31 Communicating about complex systems from the point of view of stakeholders helps them understand the consequences of their stated requirements and the design decisions based on them. Architecture gives the ability to communicate about design decisions before the system is implemented, when they are still relatively easy to adapt.
- *It helps in risk management.* Software architecture helps to reduce risks and chance of failure.[11]:18
- *It enables cost reduction.* Software architecture is a means to manage risk and costs in complex IT projects.[17]

History

The comparison between software design and (civil) architecture was first drawn in the late 1960s,[18] but the term *software architecture* became prevalent only in the beginning of the 1990s.[19] The field of computer science had encountered problems associated with complexity since its formation.[20] Earlier problems of complexity were solved by developers by choosing the right data structures, developing algorithms, and by applying the concept of separation of concerns. Although the term "software architecture" is relatively new to the industry, the fundamental principles of the field have been applied sporadically by software engineering pioneers since the mid-1980s. Early attempts to capture and explain software architecture of a system were imprecise and disorganized, often characterized by a set of box-and-line diagrams. [21]

Software architecture as a concept has its origins in the research of Edsger Dijkstra in 1968 and David Parnas in the early 1970s. These scientists emphasized that the structure of a software system matters and getting the structure right is critical. During the 1990s there was a concerted effort to define and codify fundamental aspects of the discipline, with research work concentrating on architectural styles (patterns), architecture description languages, architecture documentation, and formal methods.[22]

Research institutions have played a prominent role in furthering software architecture as a discipline. Mary Shaw and David Garlan of Carnegie Mellon wrote a book titled *Software Architecture: Perspectives on an Emerging Discipline* in 1996, which promoted software architecture concepts such as components, connectors, and styles. The University of California, Irvine's Institute for Software Research's efforts in software architecture research is directed primarily in architectural styles, architecture description languages, and dynamic architectures.

IEEE 1471-2000, *Recommended Practice for Architecture Description of Software-Intensive Systems*, was the first formal standard in the area of software architecture. It was adopted in 2007 by ISO as ISO/IEC 42010:2007. In November 2011, IEEE 1471–2000 was superseded by ISO/IEC/IEEE 42010:2011, *Systems and software engineering – Architecture description* (jointly published by IEEE and ISO).[12]

While in IEEE 1471, software architecture was about the architecture of "software-intensive systems", defined as "any system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole", the 2011 edition goes a step further by including the ISO/IEC 15288 and ISO/IEC 12207 definitions of a system, which embrace not only hardware and software, but also "humans, processes, procedures, facilities, materials and naturally occurring entities". This reflects the relationship between software architecture, enterprise architecture and solution architecture.

Architecture activities

There are many activities that a software architect performs. A software architect typically works with project managers, discusses architecturally significant requirements with stakeholders, designs a software architecture, evaluates a design, communicates with designers and stakeholders, documents the architectural design and more.[23] There are four core activities in software architecture design.[24] These core architecture activities are performed iteratively and at different stages of the initial software development life-cycle, as well as over the evolution of a system.

Architectural analysis is the process of understanding the environment in which a proposed system or systems will operate and determining the requirements for the system. The input or requirements to the analysis activity can come from any number of stakeholders and include items such as:

- what the system will do when operational (the functional requirements)
- how well the system will perform runtime non-functional requirements such as reliability, operability, performance efficiency, security, compatibility defined in ISO/IEC 25010:2011 standard[25]
- development-time non-functional requirements such as maintainability and transferability defined in ISO 25010:2011 standard[25]
- business requirements and environmental contexts of a system that may change over time, such as legal, social, financial, competitive, and technology concerns[26]

The outputs of the analysis activity are those requirements that have a measurable impact on a software system's architecture, called architecturally significant requirements.[27]

Architectural synthesis or design is the process of creating an architecture. Given the architecturally significant requirements determined by the analysis, the current state of the design and the results of any evaluation activities, the design is created and improved.[24][4]:311–326

Architecture evaluation is the process of determining how well the current design or a portion of it satisfies the requirements derived during analysis. An evaluation can occur whenever an architect is considering a design decision, it can occur after some portion of the design has been completed, it can occur after the final design has been completed or it can occur after the system has been constructed. Some of the available software architecture evaluation techniques include Architecture Tradeoff Analysis Method (ATAM) and TARA.[28] Frameworks for comparing the techniques are discussed in frameworks such as *SARA Report*[16] and *Architecture Reviews: Practice and Experience*.[29]

Architecture evolution is the process of maintaining and adapting an existing software architecture to meet changes in requirements and environment. As software architecture provides a fundamental structure of a software system, its evolution and maintenance would necessarily impact its fundamental structure. As such, architecture evolution is concerned with adding new functionality as well as maintaining existing functionality and system behavior.

Architecture requires critical supporting activities. These supporting activities take place throughout the core software architecture process. They include knowledge management and communication, design reasoning and decision making, and documentation.

Architecture supporting activities

Software architecture supporting activities are carried out during core software architecture activities. These supporting activities assist a software architect to carry out analysis, synthesis, evaluation, and evolution. For instance, an architect has to gather knowledge, make decisions and document during the analysis phase.

- **Knowledge management and communication** is the act of exploring and managing knowledge that is essential to designing a software architecture. A software architect does not work in isolation. They get inputs, functional and non-functional requirements and design contexts, from various stakeholders; and provides outputs to stakeholders. Software architecture knowledge is often tacit and is retained in the heads of stakeholders. Software architecture knowledge management activity is about finding, communicating, and retaining knowledge. As software architecture design issues are intricate and interdependent, a knowledge gap in design reasoning can lead to incorrect software architecture design.[23][30] Examples of knowledge management and communication activities include searching for design patterns, prototyping, asking experienced developers and architects, evaluating the designs of similar systems, sharing knowledge with other designers and stakeholders, and documenting experience in a wiki page.

- **Design reasoning and decision making** is the activity of evaluating design decisions. This activity is fundamental to all three core software architecture activities.[9][31] It entails gathering and associating decision contexts, formulating design decision problems, finding solution options and evaluating tradeoffs before making decisions. This process occurs at different levels of decision granularity while evaluating significant architectural requirements and software architecture decisions, and software architecture analysis, synthesis, and evaluation. Examples of reasoning activities include understanding the impacts of a requirement or a design on quality attributes, questioning the issues that a design might cause, assessing possible solution options, and evaluating the tradeoffs between solutions.
- **Documentation** is the act of recording the design generated during the software architecture process. A system design is described using several views that frequently include a static view showing the code structure of the system, a dynamic view showing the actions of the system during execution, and a deployment view showing how a system is placed on hardware for execution. Kruchten's 4+1 view suggests a description of commonly used views for documenting software architecture;[32] Documenting Software Architectures: Views and Beyond has descriptions of the kinds of notations that could be used within the view description.[1] Examples of documentation activities are writing a specification, recording a system design model, documenting a design rationale, developing a viewpoint, documenting views.

Software architecture topics

Software architecture description

Software architecture description involves the principles and practices of modeling and representing architectures, using mechanisms such as: architecture description languages, architecture viewpoints, and architecture frameworks.

Architecture description languages

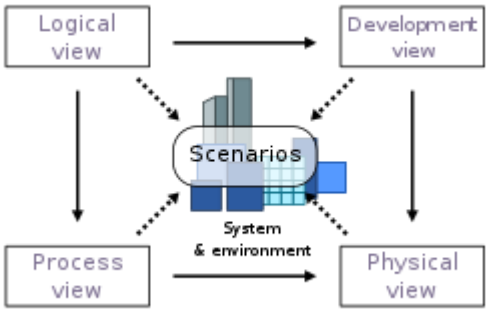
An architecture description language (ADL) is any means of expression used to describe a software architecture (ISO/IEC/IEEE 42010). Many special-purpose ADLs have been developed since the 1990s, including AADL (SAE standard), Wright (developed by Carnegie Mellon), Acme (developed by Carnegie Mellon), xADL (developed by UCI), Darwin (developed by Imperial College London), DAOP-ADL (developed by University of Málaga), SBC-ADL (developed by National Sun Yat-Sen University), and ByADL (University of L'Aquila, Italy).

Architecture viewpoints

Software architecture descriptions are commonly organized into views, which are analogous to the different types of blueprints made in building architecture. Each view addresses a set of system concerns, following the conventions of its *viewpoint*, where a viewpoint is a specification that describes the notations, modeling, and analysis techniques to use in a view that express the architecture in question from the perspective of a given set of stakeholders and their concerns (ISO/IEC/IEEE 42010). The viewpoint specifies not only the concerns framed (i.e., to be addressed) but the presentation, model kinds used, conventions used and any consistency (correspondence) rules to keep a view consistent with other views.

Architecture frameworks

An architecture framework captures the "conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders" (ISO/IEC/IEEE 42010). A framework is usually implemented in terms of one or more viewpoints or ADLs.



4+1 Architectural View Model.

Architectural styles and patterns

An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context. Architectural patterns are often documented as software design patterns.

Following traditional building architecture, a 'software architectural style' is a specific method of construction, characterized by the features that make it notable" (architectural style).

> *An architectural style defines: a family of systems in terms of a pattern of structural organization; a vocabulary of components and connectors, with constraints on how they can be combined.*[33]

> *Architectural styles are reusable 'packages' of design decisions and constraints that are applied to an architecture to induce chosen desirable qualities.*[34]

There are many recognized architectural patterns and styles, among them:

- Blackboard
- Client-server (2-tier, 3-tier, n-tier, cloud computing exhibit this style)
- Component-based
- Data-centric
- Event-driven (or implicit invocation)
- Layered (or multilayered architecture)
- Microservices architecture
- Monolithic application
- Peer-to-peer (P2P)
- Pipes and filters
- Plug-ins
- Representational state transfer (REST)
- Rule-based
- Service-oriented
- Shared nothing architecture
- Space-based architecture

Some treat architectural patterns and architectural styles as the same,[35] some treat styles as specializations of patterns. What they have in common is both patterns and styles are idioms for architects to use, they "provide a common language"[35] or "vocabulary"[33] with which to describe classes of systems.

Software architecture and agile development

There are also concerns that software architecture leads to too much Big Design Up Front, especially among proponents of agile software development. A number of methods have been developed to balance the trade-offs of up-front design and agility,[36] including the agile method DSDM which mandates a "Foundations" phase during which "just enough" architectural foundations are laid. IEEE Software devoted a special issue[37] to the interaction between agility and architecture.

Software architecture erosion

Software architecture erosion (or "decay") refers to the gap observed between the planned and actual architecture of a software system as realized in its implementation.[38] Software architecture erosion occurs when implementation decisions either do not fully achieve the architecture-as-planned or otherwise violate constraints or principles of that architecture.[2] The gap between planned and actual architectures is sometimes understood in terms of the notion of technical debt.

As an example, consider a strictly layered system, where each layer can only use services provided by the layer immediately below it. Any source code component that does not observe this constraint represents an architecture violation. If not corrected, such violations can transform the architecture into a monolithic block, with adverse effects on understandability, maintainability, and evolvability.

Various approaches have been proposed to address erosion. "These approaches, which include tools, techniques, and processes, are primarily classified into three general categories that attempt to minimize, prevent and repair architecture erosion. Within these broad categories, each approach is further broken down reflecting the high-level strategies adopted to tackle erosion. These are process-oriented architecture conformance, architecture evolution management, architecture design enforcement, architecture to implementation linkage, self-adaptation and architecture restoration techniques consisting of recovery, discovery, and reconciliation."[39]

There are two major techniques to detect architectural violations: reflexion models and domain-specific languages. Reflexion model (RM) techniques compare a high-level model provided by the system's architects with the source code implementation. There are also domain-specific languages with a focus on specifying and checking architectural constraints.

Software architecture recovery

Software architecture recovery (or reconstruction, or reverse engineering) includes the methods, techniques, and processes to uncover a software system's architecture from available information, including its implementation and documentation. Architecture recovery is often necessary to make informed decisions in the face of obsolete or out-of-date documentation and architecture erosion: implementation and maintenance decisions diverging from the envisioned architecture.[40] Practices exist to recover software architecture as Static program analysis. This is a part of subjects covered by the Software intelligence practice.

Related fields

Design

Architecture is design but not all design is architectural.[1] In practice, the architect is the one who draws the line between software architecture (architectural design) and detailed design (non-architectural design). There are no rules or guidelines that fit all cases, although there have been attempts to formalize the distinction. According to the *Intension/Locality Hypothesis*,[41] the distinction between architectural and detailed design is defined by the *Locality Criterion*,[41] according to which a statement about software design is non-local (architectural) if and only if a program that satisfies it can be expanded into a program that does not. For example, the client–server style is architectural (strategic) because a program that is built on this principle can be expanded into a program that is not client–server—for example, by adding peer-to-peer nodes.

Requirements engineering

Requirements engineering and software architecture can be seen as complementary approaches: while software architecture targets the 'solution space' or the 'how', requirements engineering addresses the 'problem space' or the 'what'.[42] Requirements engineering entails the elicitation, negotiation, specification, validation, documentation and management of requirements. Both requirements engineering and software architecture revolve around stakeholder concerns, needs and wishes.

There is considerable overlap between requirements engineering and software architecture, as evidenced for example by a study into five industrial software architecture methods that concludes that *"the inputs (goals, constrains, etc.) are usually ill-defined, and only get discovered or better understood as the architecture starts to emerge"* and that while *"most architectural concerns are expressed as requirements on the system, they can also include mandated design decisions"*.[24] In short, the choice of required behavior given a particular problem impacts the architecture of the solution that addresses that problem, while at the same time the architectural design may impact the problem and introduce new requirements.[43] Approaches such as the Twin Peaks model[44] aim to exploit the synergistic relation between requirements and architecture.

Other types of 'architecture'

Computer architecture

Computer architecture targets the internal structure of a computer system, in terms of collaborating hardware components such as the CPU – or processor – the bus and the memory.

Systems architecture

The term systems architecture has originally been applied to the architecture of systems that consists of both hardware and software. The main concern addressed by the systems architecture is then the integration of software and hardware in a complete, correctly working device. In another common – much broader – meaning, the term applies to the architecture of any complex system which may be of technical, sociotechnical or social nature.

Enterprise architecture

The goal of enterprise architecture is to "translate business vision and strategy into effective enterprise".[45] Enterprise architecture frameworks, such as TOGAF and the Zachman Framework, usually distinguish between different enterprise architecture layers. Although terminology differs from framework to framework, many include at least a distinction between a *business layer*, an *application* (or *information*) *layer*, and a *technology layer*. Enterprise architecture addresses among others the alignment between these layers, usually in a top-down approach.

See also

- Architectural pattern (computer science)
- Anti-pattern
- Attribute-driven design
- Computer architecture
- Distributed Data Management Architecture
- Distributed Relational Database Architecture
- Systems architecture
- Systems design
- Software Architecture Analysis Method
- Time-triggered system

References

1. Clements, Paul; Felix Bachmann; Len Bass; David Garlan; James Ivers; Reed Little; Paulo Merson; Robert Nord; Judith Stafford (2010). *Documenting Software Architectures: Views and Beyond, Second Edition*. Boston: Addison-Wesley. ISBN 978-0-321-55268-6.
2. Perry, D. E.; Wolf, A. L. (1992). "Foundations for the study of software architecture" (http://users.ece.utexas.edu/~perry/work/papers/swa-sen.pdf) (PDF). *ACM SIGSOFT Software Engineering Notes*. **17** (4): 40. CiteSeerX 10.1.1.40.5174 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.5174). doi:10.1145/141874.141884 (https://doi.org/10.1145%2F141874.141884).
3. "Software Architecture" (https://www.sei.cmu.edu/research-capabilities/all-work/display.cfm?customel_datapageid_4050=21328). *www.sei.cmu.edu*. Retrieved 2018-07-23.
4. Bass, Len; Paul Clements; Rick Kazman (2012). *Software Architecture in Practice, Third Edition*. Boston: Addison-Wesley. ISBN 978-0-321-81573-6.
5. SEI (2006). "How do you define Software Architecture?" (http://www.sei.cmu.edu/architecture/start/glossary/definition-form.cfm). Retrieved 2012-09-12.
6. Garlan & Shaw (1994). "An Introduction to Software Architecture" (http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf) (PDF). Retrieved 2012-09-13.
7. Fowler, M. (2003). "Design – Who needs an architect?". *IEEE Software*. **20** (5): 11–44. doi:10.1109/MS.2003.1231144 (https://doi.org/10.1109%2FMS.2003.1231144).
8. ISO/IEC/IEEE 42010: Defining "architecture" (http://www.iso-architecture.org/42010/defining-architecture.html). Iso-architecture.org. Retrieved on 2013-07-21.

9. Jansen, A.; Bosch, J. (2005). "Software Architecture as a Set of Architectural Design Decisions". *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. p. 109. CiteSeerX 10.1.1.60.8680 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.8680). doi:10.1109/WICSA.2005.61 (https://doi.org/10.1109%2FWICSA.2005.61). ISBN 978-0-7695-2548-8.
10. Ali Babar, Muhammad; Dingsoyr, Torgeir; Lago, Patricia; van Vliet, Hans (2009). *Software Architecture Knowledge Management*. Dordrecht Heidelberg London New York: Springer. ISBN 978-3-642-02373-6.
11. George Fairbanks (2010). *Just Enough Software Architecture*. Marshall & Brainerd.
12. ISO/IEC/IEEE (2011). "ISO/IEC/IEEE 42010:2011 Systems and software engineering – Architecture description" (http://www.iso.org/iso/catalogue_detail.htm?csnumber=50508). Retrieved 2012-09-12.
13. Muller, Gerrit (August 20, 2007). "A Reference Architecture Primer" (http://www.gaudisite.nl/ReferenceArchitecturePrimerPaper.pdf) (PDF). *Gaudi site*. Retrieved November 13, 2015.
14. Angelov, Samuil; Grefen, Paul; Greefhorst, Danny (2009). "A Classification of Software Reference Architectures: Analyzing Their Success and Effectiveness". *Proc. Of WICSA/ECSA 2009*: 141–150. CiteSeerX 10.1.1.525.7208 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.525.7208). doi:10.1109/WICSA.2009.5290800 (https://doi.org/10.1109%2FWICSA.2009.5290800). ISBN 978-1-4244-4984-2.
15. Brooks, Jr., Frederick P. (1975). *The Mythical Man-Month – Essays on Software Engineering*. Addison-Wesley. ISBN 978-0-201-00650-6.
16. Obbink, H.; Kruchten, P.; Kozaczynski, W.; Postema, H.; Ran, A.; Dominick, L.; Kazman, R.; Hilliard, R.; Tracz, W.; Kahane, E. (Feb 6, 2002). "Software Architecture Review and Assessment (SARA) Report" (https://pkruchten.files.wordpress.com/2011/09/sarav1.pdf) (PDF). Retrieved November 1, 2015.
17. Poort, Eltjo; van Vliet, Hans (September 2012). "RCDA: Architecting as a risk- and cost management discipline". *Journal of Systems and Software*. **85** (9): 1995–2013. doi:10.1016/j.jss.2012.03.071 (https://doi.org/10.1016%2Fj.jss.2012.03.071).
18. P. Naur; B. Randell, eds. (1969). "Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7–11 Oct. 1968" (http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF) (PDF). Brussels: NATO, Scientific Affairs Division. Retrieved 2012-11-16.
19. P. Kruchten; H. Obbink; J. Stafford (2006). "The past, present and future of software architecture". *IEEE Software*. **23** (2): 22. doi:10.1109/MS.2006.59 (https://doi.org/10.1109%2FMS.2006.59).
20. University of Waterloo (2006). "A Very Brief History of Computer Science" (http://www.cs.uwaterloo.ca/~shallit/Courses/134/history.html). Retrieved 2006-09-23.
21. IEEE Transactions on Software Engineering (2006). "Introduction to the Special Issue on Software Architecture". doi:10.1109/TSE.1995.10003 (https://doi.org/10.1109%2FTSE.1995.10003).
22. Garlan & Shaw (1994). "An Introduction to Software Architecture" (http://www.cs.cmu.edu/afs/cs/project/able/ftp/intro_softarch/intro_softarch.pdf) (PDF). Retrieved 2006-09-25.
23. Kruchten, P. (2008). "What do software architects really do?". *Journal of Systems and Software*. **81** (12): 2413–2416. doi:10.1016/j.jss.2008.08.025 (https://doi.org/10.1016%2Fj.jss.2008.08.025).
24. Christine Hofmeister; Philippe Kruchten; Robert L. Nord; Henk Obbink; Alexander Ran; Pierre America (2007). "A general model of software architecture design derived from five industrial approaches". *Journal of Systems and Software*. **80** (1): 106–126. doi:10.1016/j.jss.2006.05.024 (https://doi.org/10.1016%2Fj.jss.2006.05.024).
25. ISO/IEC (2011). "ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models" (http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=35733). Retrieved 2012-10-08.
26. Osterwalder and Pigneur (2004). "An Ontology for e-Business Models" (https://pdfs.semanticscholar.org/8513/9070e23b0b3278d73ea51b873acd99352e9c.pdf) (PDF): 65–97. CiteSeerX 10.1.1.9.6922 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.9.6922).

27. Chen, Lianping; Ali Babar, Muhammad; Nuseibeh, Bashar (2013). "Characterizing Architecturally Significant Requirements". *IEEE Software*. **30** (2): 38–45. doi:10.1109/MS.2012.174 (https://doi.org/10.1109%2FMS.2012.174). hdl:10344/3061 (https://hdl.handle.net/10344%2F3061).
28. Woods, E. (2012). "Industrial architectural assessment using TARA". *Journal of Systems and Software*. **85** (9): 2034–2047. doi:10.1016/j.jss.2012.04.055 (https://doi.org/10.1016%2Fj.jss.2012.04.055).
29. Maranzano, J. F.; Rozsypal, S. A.; Zimmerman, G. H.; Warnken, G. W.; Wirth, P. E.; Weiss, D. M. (2005). "Architecture Reviews: Practice and Experience". *IEEE Software*. **22** (2): 34. doi:10.1109/MS.2005.28 (https://doi.org/10.1109%2FMS.2005.28).
30. Babar, M.A.; Dingsøyr, T.; Lago, P.; Vliet, H. van (2009). *Software Architecture Knowledge Management:Theory and Practice (eds.), First Edition*. Springer. ISBN 978-3-642-02373-6.
31. Tang, A.; Han, J.; Vasa, R. (2009). "Software Architecture Design Reasoning: A Case for Improved Methodology Support". *IEEE Software*. **26** (2): 43. doi:10.1109/MS.2009.46 (https://doi.org/10.1109%2FMS.2009.46).
32. Kruchten, Philippe (1995). "Architectural Blueprints – The '4+1' View Model of Software Architecture" (http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf) (PDF). *IEEE Software*. **12** (6): 42–50. doi:10.1109/52.469759 (https://doi.org/10.1109%2F52.469759).
33. Shaw, Mary; Garlan, David (1996). *Software architecture: perspectives on an emerging discipline*. Prentice Hall. ISBN 978-0-13-182957-2.
34. UCI Software Architecture Research – UCI Software Architecture Research: Architectural Styles (http://www.isr.uci.edu/architecture/styles.html). Isr.uci.edu. Retrieved on 2013-07-21.
35. Chapter 3: Architectural Patterns and Styles (http://msdn.microsoft.com/en-us/library/ee658117.aspx). Msdn.microsoft.com. Retrieved on 2013-07-21.
36. Boehm, Barry; Turner, Richard (2004). *Balancing Agility and Discipline*. Addison-Wesley. ISBN 978-0-321-18612-6.
37. "IEEE Software Special Issue on Agility and Architecture" (http://www.computer.org/portal/web/computingnow/archive/april2010). April 2010. Retrieved 14 September 2012.
38. Terra, R., M.T. Valente, K. Czarnecki, and R.S. Bigonha, "Recommending Refactorings to Reverse Software Architecture Erosion", 16th European Conference on Software Maintenance and Reengineering, 2012. http://gsd.uwaterloo.ca/sites/default/files/Full%20Text.pdf
39. de Silva, L.; Balasubramaniam, D. (2012). "Controlling software architecture erosion: A survey". *Journal of Systems and Software*. **85** (1): 132–151. doi:10.1016/j.jss.2011.07.036 (https://doi.org/10.1016%2Fj.jss.2011.07.036).
40. Lungu, M. "Software architecture recovery", University of Lugano, 2008. http://www.slideshare.net/mircea.lungu/software-architecture-recovery-in-five-questions-presentation
41. Amnon H. Eden; Rick Kazman (2003). "Architecture Design Implementation" (https://web.archive.org/web/20070928035606/http://eden-study.org/articles/2003/icse03.pdf) (PDF). Archived from the original (http://www.eden-study.org/articles/2003/icse03.pdf) (PDF) on 2007-09-28.
42. C. Shekaran; D. Garlan; M. Jackson; N.R. Mead; C. Potts; H.B. Reubenstein (1994). "The role of software architecture in requirements engineering". *Proceedings of IEEE International Conference on Requirements Engineering*: 239–245. doi:10.1109/ICRE.1994.292379 (https://doi.org/10.1109%2FICRE.1994.292379). ISBN 978-0-8186-5480-0.
43. Remco C. de Boer, Hans van Vliet (2009). "On the similarity between requirements and architecture". *Journal of Systems and Software*. **82** (3): 544–550. CiteSeerX 10.1.1.415.6023 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.415.6023). doi:10.1016/j.jss.2008.11.185 (https://doi.org/10.1016%2Fj.jss.2008.11.185).
44. Bashar Nuseibeh (2001). "Weaving together requirements and architectures" (http://oro.open.ac.uk/2213/1/00910904.pdf) (PDF). *Computer*. **34** (3): 115–119. doi:10.1109/2.910904 (https://doi.org/10.1109%2F2.910904).
45. Definition of Enterprise Architecture, Gartner (http://www.gartner.com/technology/it-glossary/enterprise-architecture.jsp)

Further reading

- Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Robert Nord, Judith Stafford: *Documenting Software Architectures: Views and Beyond, Second Edition*. Addison-Wesley, 2010, ISBN 0-321-55268-7.

  > This book describes what is software architecture and shows how to document it in multiple views, using UML and other notations. It also explains how to complement the architecture views with behavior, software interface, and rationale documentation. Accompanying the book is a wiki that contains an example of software architecture documentation (https://wiki.sei.cmu.edu/sad/index.php/The_Adventure_Builder_SAD).

- Len Bass, Paul Clements, Rick Kazman: *Software Architecture in Practice, Third Edition*. Addison Wesley, 2012, ISBN 0-321-81573-4 (This book, now in third edition, eloquently covers the fundamental concepts of the discipline. The theme is centered on achieving quality attributes of a system.)
- Amnon H. Eden, Rick Kazman. *Architecture, Design, Implementation.* (https://web.archive.org/web/20070928035606/http://eden-study.org/articles/2003/icse03.pdf) On the distinction between architectural design and detailed design.
- Garzás, Javier; Piattini, Mario (2005). "An ontology for micro-architectural design knowledge". *IEEE Software*. **22** (2): 28–33. doi:10.1109/MS.2005.26 (https://doi.org/10.1109%2FMS.2005.26).
- Kruchten, Philippe (1995). "Architectural Blueprints – The '4+1' View Model of Software Architecture" (http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/Pbk4p1.pdf) (PDF). *IEEE Software*. **12** (6): 42–50. doi:10.1109/52.469759 (https://doi.org/10.1109%2F52.469759).
- Shan, Tony; Hua, Winnie (October 2006). "Solution Architecting Mechanism". *Proceedings of the 10th IEEE International EDOC Enterprise Computing Conference*: 23–32. doi:10.1109/EDOC.2006.54 (https://doi.org/10.1109%2FEDOC.2006.54). ISBN 978-0-7695-2558-7.
- Martin Fowler (with Ralph Johnson) Who Needs an Architect? (http://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf) IEEE Software, Jul/Aug 2003
- Bell, Michael (2008). *Service-Oriented Modeling: Service Analysis, Design, and Architecture*. Wiley. ASIN 0470141115 (https://www.amazon.com/dp/0470141115).

External links

- Explanation on IBM Developerworks (http://www.ibm.com/developerworks/rational/library/feb06/eeles/)
- Collection of software architecture definitions (http://www.sei.cmu.edu/architecture/start/definitions.cfm) at Software Engineering Institute (SEI), Carnegie Mellon University (CMU)
- International Association of IT Architects (IASA Global) (http://www.iasaglobal.org/), formerly known as the International Association for Software Architects (IASA)
- SoftwareArchitecturePortal.org (http://www.softwarearchitectureportal.org/) – website of IFIP Working Group 2.10 on Software Architecture
- SoftwareArchitectures.com (http://www.softwarearchitectures.com/) – independent resource of information on the discipline
- Software Architecture (http://www.ics.uci.edu/~fielding/pubs/dissertation/software_arch.htm), chapter 1 of Roy Fielding's REST dissertation
- When Good Architecture Goes Bad (http://www.methodsandtools.com/archive/archive.php?id=85)
- The Spiral Architecture Driven Development (http://sadd.codeplex.com) – the SDLC based on Spiral model is to reduce the risks of ineffective architecture
- Software Architecture Real Life Case Studies (https://www.infoq.com/architecture)
