SOFTWARE ARCHITECTURE AND SOFTWARE PROCESSES

BY PANKAJ KAMTHAN

1. INTRODUCTION

The things we have to learn **before** we do them, we learn by doing them.
— Aristotle

This document explores the relationship between software processes and software architecture. In doing so, it discusses certain basic concepts of software process engineering.

For a software designer, a software system's architecture is both a **destination** (that is, product) and a **journey** (that is, process) [Booch, 2007].

In software engineering, the approach to, and the development of, any artifact should be **systematic and disciplined**. This is accomplished by means of a process.

2. PROCESS GALORE

In a software project, there can be **several types of processes**, as illustrated in Figure 1. The following are some common processes:

- Quality Management Process
- Risk Management Process
- Configuration Management Process
- Measurement Process
- Risk Management Process
- **Software Architectural Design Process** (Clause 7.1.3 of the ISO/IEC 12207 Standard)
- Software Verification Process
- Software Validation Process

The adoption of a process **depends** on the organization responsible for the software project. Therefore, with respect to the number and types of processes being used in an organization, there can be **variations** across software projects.

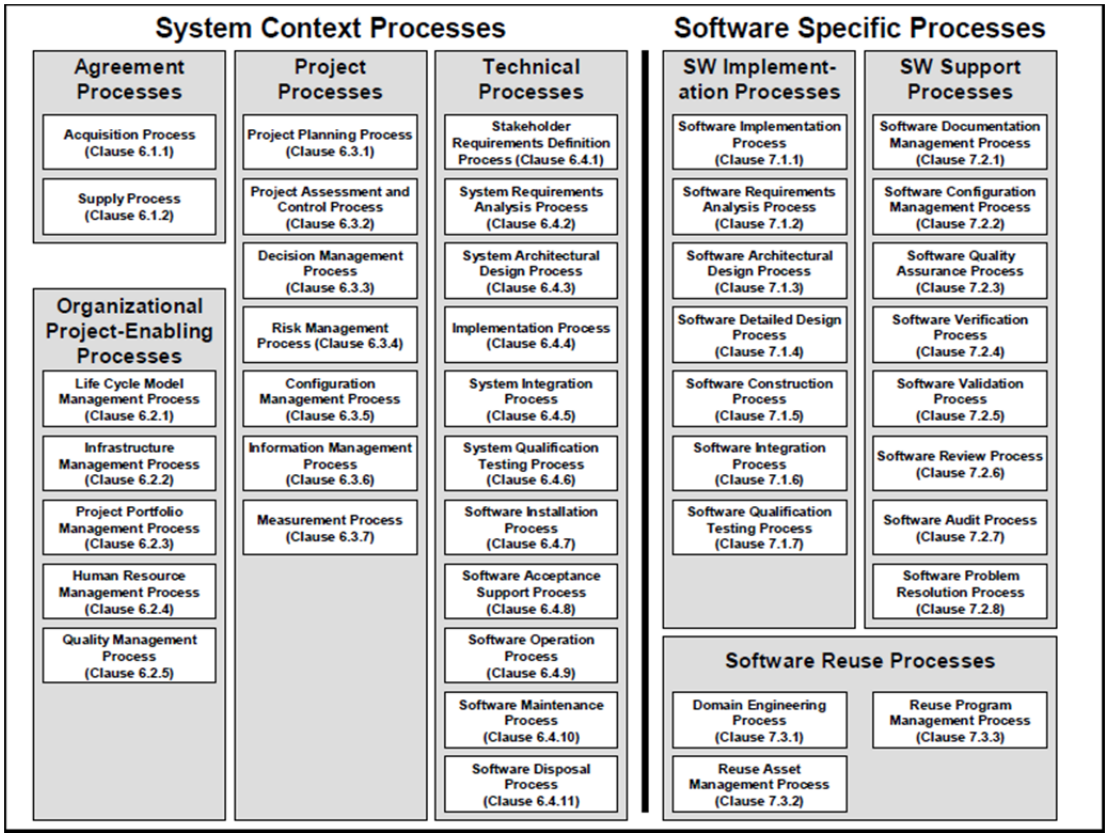


Figure 1. A collection of processes and process groups. (Source: [ISO/IEC, 2008].)

3. DEFINITION OF PROCESS

There are **different meanings** of the term 'process', essentially **unrelated** to each other. For example, process in the sense of **operating systems** is **not** the same as process in the sense of **software engineering**.

There are a number of definitions of process, including the following.

DEFINITION 1

The next definition of process is from the **viewpoint of time**.

Definition [Process] [IEEE, 1990]. A sequence of steps performed for a given purpose; for example, such as, software development process.

DEFINITION 2

Definition [Activity] [ISO/IEC/IEEE, 2010].
(1) A defined body of work to be performed, including its **required input and output information**.
(2) [A] set of **cohesive tasks** of a process.

The next definition of process is from the **viewpoint of activity**.

Definition [Process] [ISO/IEC/IEEE, 2010]. [A] set of interrelated or interacting activities which transforms inputs into outputs.

OBSERVATIONS

- The term **"activities"** covers use of resources.
- A process may have **multiple starting points** and **multiple end points**.
- The prescribed manner may be a **partially ordered temporal sequence**.

4. CLASSIFICATION OF PROCESSES

There are a number of ways of classifying processes.

For example, a process could be one or more of the following: **sequential or parallel, linear or non-linear, predictive or adaptive, rigid or agile**, and so on.

5. DEFINITION OF SOFTWARE PROCESS AND RELATED TERMS

There are a number of definitions of **software development process (or, equivalently, software process)**, including the following:

Definition [Software Development Process] [ISO/IEC/IEEE, 2010]. The process by which **user needs** are translated into a software product.

OBSERVATIONS

- The process involves **translating** user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use.

- These activities may **overlap** or be performed **iteratively**.

CONCLUSIONS

From the previous definition, it can be concluded that there are two **characteristics of a software development process**:

- Activities
- Artifacts

Furthermore, if the activities overlap, they may be **interspersed (that is, interleave)** or could be performed in **parallel**. The artifacts output from one activity could be an input to another activity.

RELATED TERMS

Definition [Software Development Life Cycle] [IEEE, 1990]. The period of time that begins with the **decision to develop** a software product and ends when the software is **delivered**.

Definition [Software Life Cycle] [IEEE, 1990]. The period of time that begins when a software product is **conceived** and ends when the software is **no longer available for use**. This is illustrated in Figure 2.

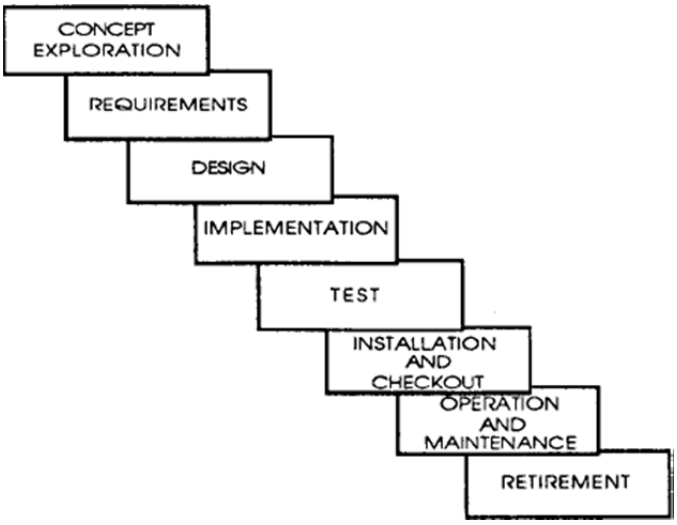


Figure 2. A software life cycle. (Source: [IEEE, 1990].)

OBSERVATIONS

The terms ‘software development life cycle’ and ‘software life cycle’ are **not synonymous**, as indicated by their respective definitions.

In other words, software development process, software development life cycle, and software life cycle have **similarities, as well as differences**.

There are a number of definitions of methodology, including the following:

Definition [Methodology] [IEEE, 1995]. A comprehensive, integrated series of techniques or methods creating a general systems theory of how a class of thought-intensive work ought to be performed.

Definition [Methodology] [ISO/IEC/IEEE, 2010]. [A] system of practices, techniques, procedures, and rules used by those who work in a discipline.
Note: A methodology **specifies** the **process** to be executed, usually as a set of related activities, tasks and/or techniques, **together** with the work products that must be manipulated (created, used or changed) at each moment and by whom, possibly including models, documents and other inputs and outputs.

OBSERVATIONS

- It follows from their respective definitions that a **methodology, as a concept, is philosophical**; a **process, as a concept, is technical**.

- A methodology is **more general than a process** in the sense that a methodology **subsumes** a process.

There are a number of software development methodologies, and each methodology is usually equipped with a process model.

6. INCREMENTAL VERSUS ITERATIVE

The idea of iterative and incremental development is related to the notion of **'piecemeal growth'** [Alexander, 1964]. It is in **contrast** to **Big Design Up Front (BDUF)** (also known as **upfront design**) [Fairbanks, 2010, Section 5.5].

There are **differences** between a (software development) process being incremental and being iterative.

INCREMENTAL

Let **Product Part 1**, **Product Part 2**, …, **Product Part n** denote parts of a product for some software project. Then, as suggested by Figure 3:

$$\textbf{Product Part 1} \subset \textbf{Product Part 2} \subset \ldots \subset \textbf{Product Part n}$$

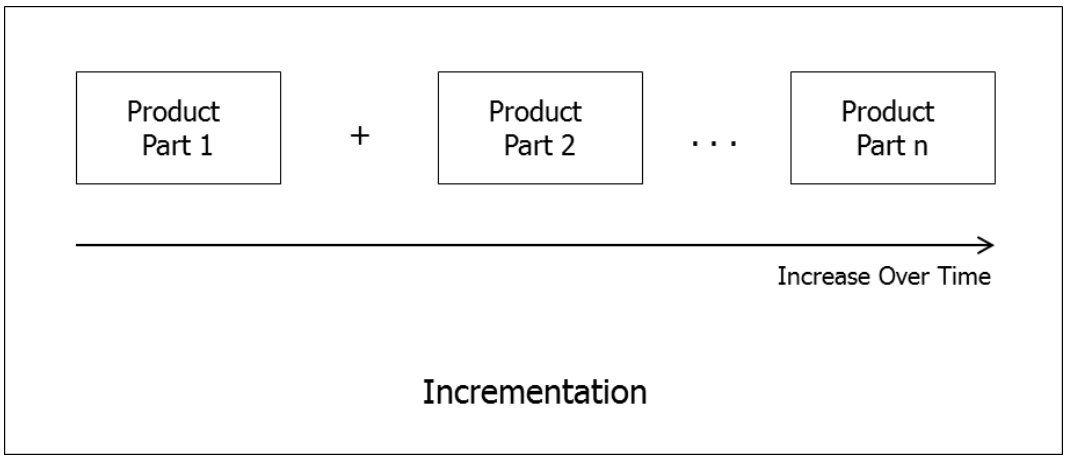


Figure 3. An abstract incremental process.

ITERATIVE

The most successful men in the end are those whose success is the result of steady accretion.
— Alexander Graham Bell

Let **Product Part i** be some software engineering product. Then, as suggested by Figure 4, **Product Part i** can be revisited and improved over time.

The presence of color (grey) indicates the presence of **undesirables** related to product quality that are expected to **reduce** over time. (The undesirables could, for example, be 'smells' or errors.)

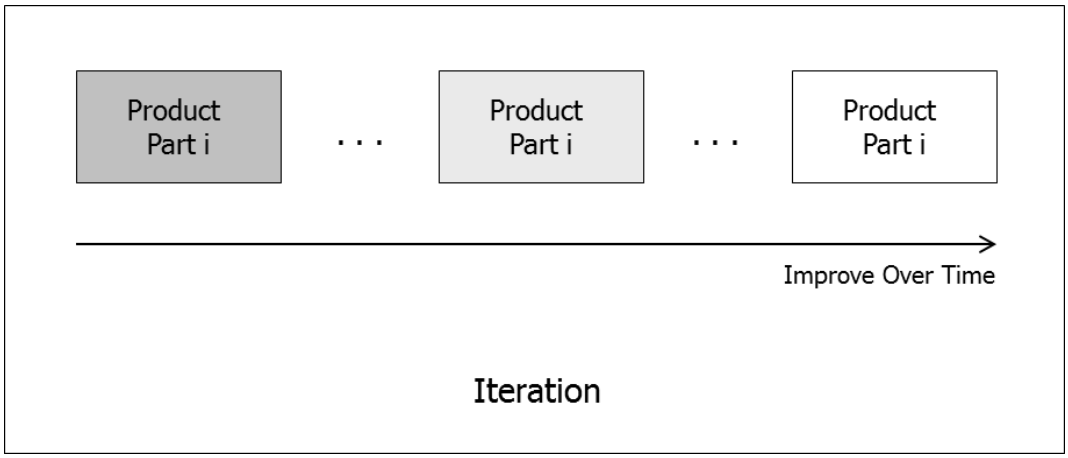


Figure 4. An abstract iterative process.

However, in certain processes, the terms 'iterative' and 'incremental' have been **merged** [Cockburn, 2008]. The result is that an 'iterative' process means **both** iterative and incremental.

The **reason** for this merger is the following. It is **insufficient** for a process to be **only iterative** (or **only incremental**).

7. SITUATING SOFTWARE ARCHITECTURE IN A SOFTWARE PROCESS

The **placement** of, **emphasis** on, and **treatment** of software architecture in software process models, varies.

7.1. WATERFALL MODELS AND SOFTWARE ARCHITECTURE

It seems to me that textbooks (and therefore courses) give the impression that early software development was done by a bunch of idiots getting drowned in waterfalls. The truth is that there were many very smart people around, and good practices emerged early on (although they were not employed everywhere, of course).

— Peter Grogono

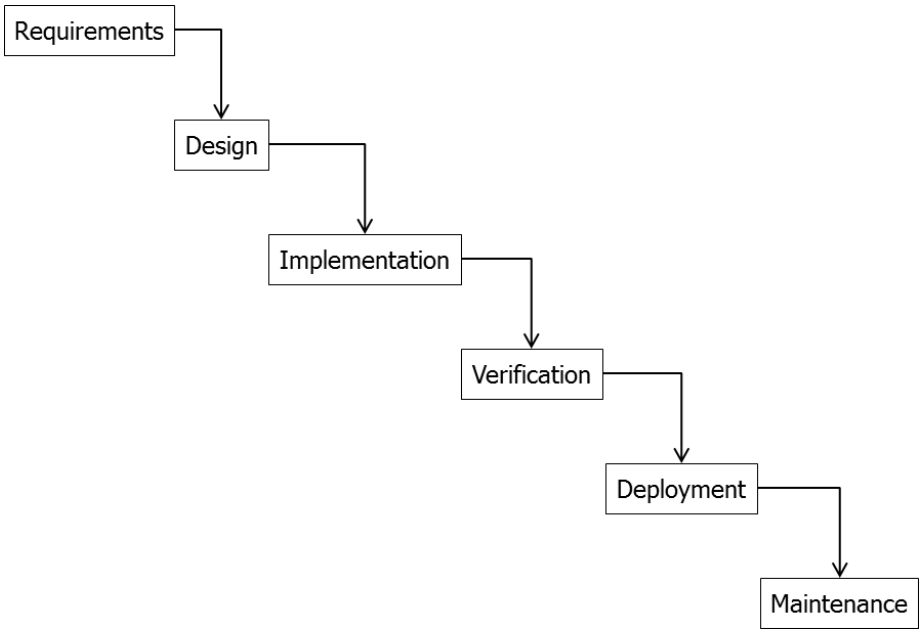
A ‘strict’ Waterfall Model, as illustrated in Figure 5(a), is considered **impractical** [Royce, 1970]. Indeed, the argument of [Royce, 1970] is that, although the use of Waterfall Model was common in ‘conventional’ engineering, it would **not** be suited for software engineering.

There are a number of **variations** of a Waterfall Model [Stephens, 2015, Chapter 12] based on (1) the number of phases, (2) the types of phases included, and (3) the topology of transitions, as illustrated in Figure 5(b).

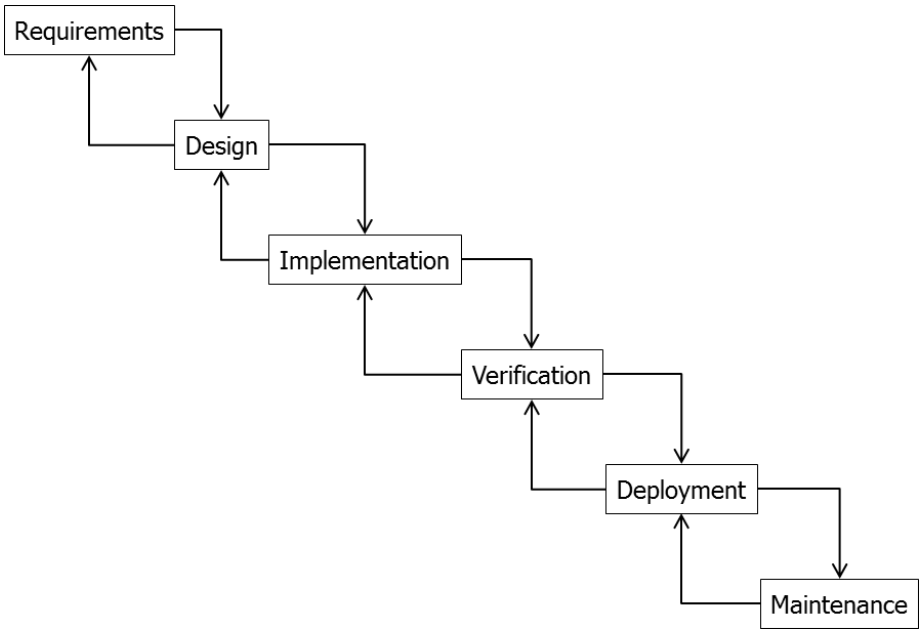
A Waterfall Model has **one dimension**.

A typical Waterfall Model has the following phases:

1. Software Requirements
2. **Software Design**
3. Software Implementation
4. Software Verification
5. Software Deployment
6. Software Maintenance



(a)



(b)

Figure 5. (a) A Waterfall Model, and (b) a Waterfall Model with Feedback.

In a Waterfall Model [Royce, 1970], software architecture design is part of the software design phase.

In this sense, software architecture is a **'bridge'** between software requirements and software implementation [Garlan, 2000]. It is important that software requirements are complete before software architecture begins [Palmquist, Lapham, Miller, Chick, Ozkaya, 2013].

The software architecture defines **constraints** on implementation. A software-intensive system's implementation is the **manifestation** of its architecture, and a system's architecture **shapes** its implementation [Booch, 2007]. It is the purpose of software verification to ensure that software implementation corresponds to software architecture.

7.2. THE UNIFIED PROCESS AND SOFTWARE ARCHITECTURE

The **Unified Software Development Process** or **Unified Process (UP)** [Jacobson, Booch, Rumbaugh, 1999] is a use-case driven, architecture-centric, risk-focused, iterative and incremental software development process framework.

- **Use-Case Driven.** UP is use-case driven in the sense that the **progress** is determined by attention to use cases.

- **Architecture-Centric.** UP is architecture-centric in the sense that architecture is at the center of all **critical decision-making**.

- **Risk-Focused.** UP is risk-focused in the sense that critical risks are addressed as early as possible: commitments are made either to avoid risk or to ameliorate risk.

- **Iterative.** UP is iterative in the sense that it allows revisitation (and revision) of an artifact.

- **Incremental.** UP is incremental in the sense that it allows development through approximations.

- **Software Development Process Framework.** UP, as a (process) **'framework'**, can be **customized** the needs of specific organizations or projects.

Figure 6 illustrates the UP model.

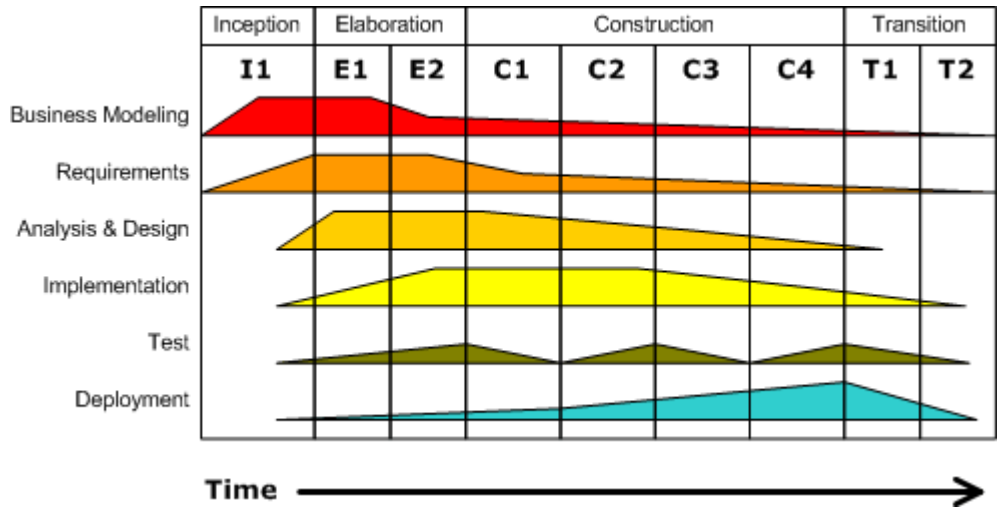


Figure 6. The Unified Process Model. (Source: Wikipedia.)

The UP model has **two dimensions**:

- **The Horizontal Axis.** This represents time and shows the life-cycle aspects (or **phases**) of the process as it unfolds.

- **The Vertical Axis.** This represents core process **disciplines** (or **workflows**) that **group activities logically and temporally**[1].

In UP, **most** of the **Design** is carried out during the **Elaboration Phase and Construction Phase**.

7.2.1. REALIZATIONS OF UP

In **practice**, it is possible to carry out UP in **different ways**, not necessarily with the same results. Figures 7-9, all from "Understanding the Unified Process (UP)[2]", illustrate realizations of UP using **linear, sequential, and iterative (and incremental) approaches**, respectively. Figure 9 is similar to Figure 6. (It is possible to replace the lifecycle instance (`:Lifecycle`) by that of a **software process instance**, and the term **'discipline'** by **workflow**.)

UP: LINEAR APPROACH

PHILOSOPHY

- This approach is **macro-focused** towards **phases** where workflows are more **discretely distributed across phases**.

- The balance between business and technology is **skewed towards management stakeholders**.

- The effort attempts to force **all use cases** through a **few workflows** in an iteration. This is often because the management stakeholders perceive virtually everything as a **business risk** that must be **controlled urgently**.

EXECUTION

- The first set of iterations focus primarily on business modeling, next set of iterations focus primarily on requirements, and so on, as evident by **short vertical lines** in the boxes in Figure 7.

---

[1] There is a **difference** between logical and temporal organization of activities. The decision that the **interviews of all relevant stakeholders of the software requirements** should be included in a single workflow is an example of **logical organization**, and that **prioritization of software requirements** can be carried out only after there is **description of (multiple) software requirements** is an example of **temporal organization**.

[2] URL: http://www.methodsandtools.com/archive/archive.php?id=32 .

- The team steadily learns more about the problem **before** learning about the solution, as the effort progresses across phases.

- The effort results in a **complete system** only at the **end** of the development cycle.

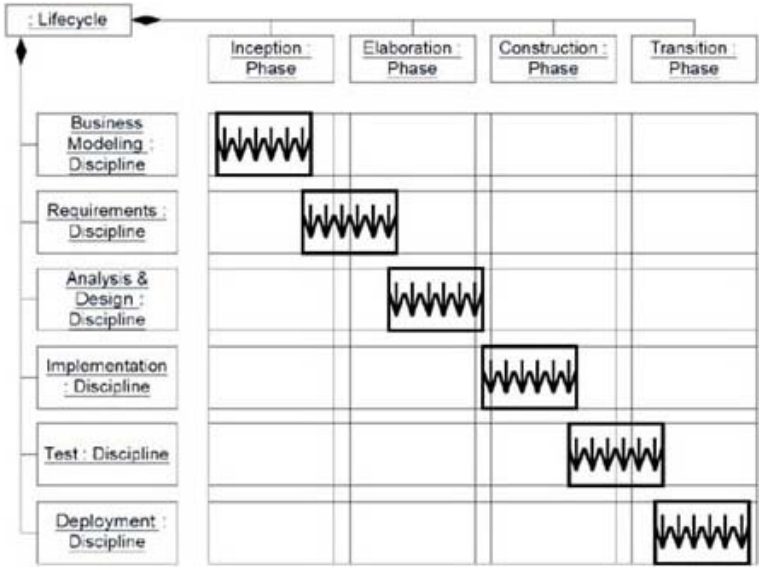


Figure 7. UP realization using a linear approach.

UP: SEQUENTIAL APPROACH

PHILOSOPHY

- This approach is **micro-focused** towards **iterations** where workflows are more **discretely distributed within iterations**.

- The balance between business and technology is **skewed by the technical stakeholders**.

- The effort attempts to force a **few use cases** through **all workflows** in an iteration. This is often because the technical stakeholders perceive virtually everything as a **technical risk** that must be **addressed immediately**.

EXECUTION

- The **use cases evolve** through each workflow in a single iteration, as evident by **long vertical lines** in the boxes in Figure 8.

- The team steadily learns more about the solution for a **limited portion** of the problem as the effort progresses across phases.

- The effort results in a system that only addresses a **subset** of the requirements, which **may or may not** be deployable.

- The effort results in a **complete system** only at the **end** of the development cycle.

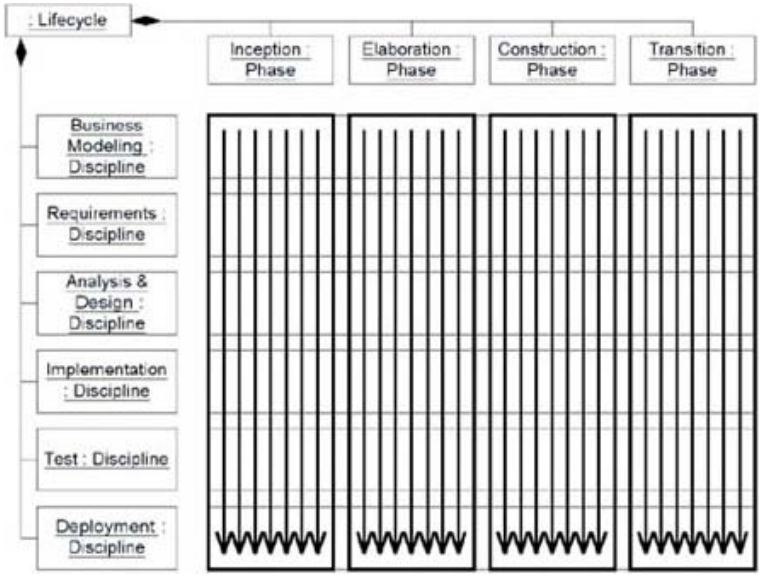


Figure 8. UP realization using a sequential approach.

UP: ITERATIVE (AND INCREMENTAL) APPROACH

PHILOSOPHY

- This approach focuses on a **stepwise refinement of knowledge throughout** the lifecycle.

- The **parallelogram** in Figure 9 reflects the **distribution of effort**. The top-left and bottom-right **corners** of the parallelogram can be **'tailored'** to suit specific projects.

- A **'pure' waterfall** results when the sides of the parallelogram are **'collapsed'** onto its main diagonal.

EXECUTION

A **linear** approach focuses on the **problem**, and a **sequential** approach focuses on the **solution**. An iterative (and incremental) approach involves using a **combination** of linear and sequential approaches.

- **Inception Phase:** The linear approaches focus on scope, and sequential approaches focus on an **architectural proof-of-concept**.

- **Elaboration Phase:** The linear approaches focus on **architectural coverage**, and sequential approaches focus on addressing **architectural risk**.

- **Construction Phase:** The sequential approaches lead to **deployment opportunities**.

- **Transition Phase:** The linear and sequential approaches focus on **system completion and project closure**.

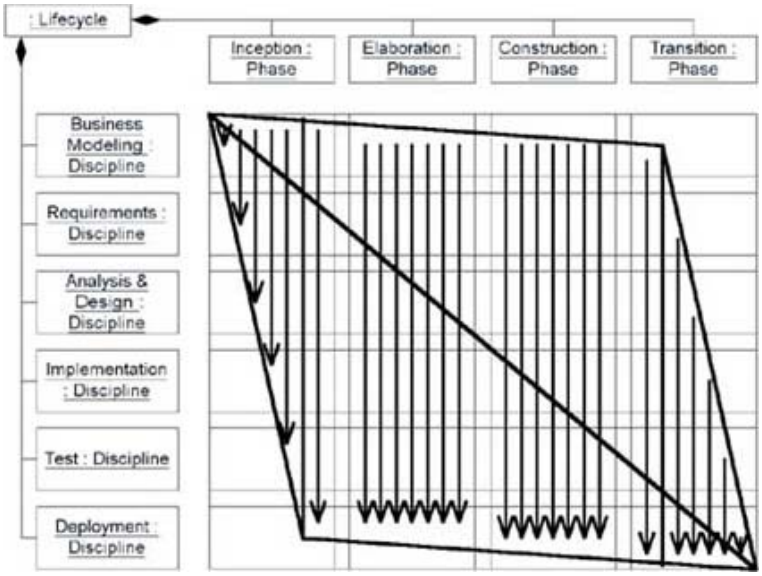


Figure 9. UP realization using an iterative approach.

REMARKS

The **Rational Unified Process (RUP)** [Kruchten, 2004] is a **customization** of the UP. The role of software architecture in the RUP has been discussed in [Kazman, Kruchten, Nord, Tomayko, 2004].

7.3. AGILE METHODOLOGIES AND SOFTWARE ARCHITECTURE

The **Agile Manifesto**[3] introduces the underlying philosophy, including values and principles, for introducing agility in software development.

A software development methodology is an **agile methodology** if it is based on the Agile Manifesto**.**

7.3.1. OPENUP

The **Open Unified Process (OpenUP)** [Kroll, MacIsaac, 2006] is an **agile customization** of the UP. Its emphasis on software architecture is therefore derived from UP.

However, being an agile methodology, OpenUP also aims to keep the use of tools and documentation to a **necessary minimum**[4].

7.3.2. CHALLENGES IN CO-EXISTENCE OF AGILITY AND ARCHITECTURE

No one would want to fly in an aircraft with flight control software that had not been rigorously planned and thoroughly analyzed.

No one would want to spend 18 months planning an e-commerce web site for their latest cell phone model, or video game, or women's shoe style (all of which are guaranteed to be badly out of fashion in 18 months).
— Rick Kazman

[3] URL: http://agilemanifesto.org/ .
[4] URL: http://epf.eclipse.org/wikis/openup/ .

The treatment of software architecture in agile methodologies[5] is currently an **active area of research** and there are a number of **challenges** [Rozanski, Woods, 2005, Chapter 7; Nord, Tomayko, 2006; Babar, 2009; Durdik, 2011; Babar, Brown, Mistrik, 2014].

There are certain **points of contention**[6] in a **reconciliation** and **co-existence** of 'conventional' approaches to software architecture and agile methodologies [Abrahamsson, Babar, Kruchten, 2010; Babar, Brown, Mistrik, 2014]:

- Face-To-Face Communication over Documentation
- Upfront Design versus Refactoring

FACE-TO-FACE COMMUNICATION VERSUS DOCUMENTATION

It is known that **agile methodologies** emphasize face-to-face communication over documentation.

However, the projects for which software architecture serves the **most important function** are **large, distributed, and long-lived**. These also happen to be the projects for which face-to-face communication is the **least practical** [Clements, Ivers, Little, Nord, Stafford, 2003].

UPFRONT DESIGN VERSUS REFACTORING

In agile methodologies, the **upfront design** and evaluation of architecture[7] is **discouraged**, and **refactoring** is **encouraged**. The idea is that the software architecture **'emerges'** over time.

However, a large-scale refactoring is either **not possible** (especially later in the development), or **not feasible** (since it is prohibitively expensive).

7.3.2.1. ON AGILE SOFTWARE ARCHITECTURE IN PRACTICE

It is important to realize that the forgoing points of contention do **not** have to be "either/or" choices with "all or nothing". In other words, for any successful project, agile or rigid, there needs to **strike a balance** between **face-to-face communication and documentation**, and between **upfront design and refactoring**.

[5] URL: http://www.sei.cmu.edu/architecture/research/archpractices/Agile-Architecting.cfm .
[6] URL: http://www.agilemodeling.com/essays/agileArchitecture.htm .
[7] This is also known as **Big Design Upfront (BDUF)** [Kruchten, Obbink, Stafford, 2006].

Indeed, it is possible to **incorporate 'conventional' practices of software architecture in agile software development**, some of which have been pointed out in [Babar, Brown, Mistrik, 2014, Foreword, Page 16, and Chapter 4]:

- **Suggestion 1:** If the software requirements are unstable, then (1) candidate software architecture could be designed in a manner that it leaves out details, and (2) agreement on the major patterns to be used could be sought.

- **Suggestion 2:** In XP, the practice of **"Onsite Customer"** could involve key stakeholders of the software architecture. For interactive software systems, such stakeholders could include **architecturally significant personas**.

- **Suggestion 3:** In Scrum, in each sprint, the practice of **"Sprint Planning"** could involve identification and prioritization **architecturally significant requirements**. (The **prioritized architecturally significant requirements** are called **architectural drivers** [Sangwan, 2015, Chapter 4].)

- **Suggestion 4:** To satisfy the practice of **"Simple Design"**, established **software architecture patterns** could be selected and followed.

- **Suggestion 5:** In Scrum, the practice of **"Sprint Review"** could incorporate **software architecture review**.

7.4. SOFTWARE DEVELOPMENT METHODOLOGIES AND SOFTWARE ARCHITECTURE INDUSTRY

In one of the analytical studies, it has been shown that the support for software architecture in several software development methodologies (specifically, **RUP, MSF, MBASE. and RUP-SOA**) is, in general, strong, but the adoption of underlying software architecture practices in industry is weak [Reyes-Delgado, Mora, Duran-Limon, Rodríguez-Martínez, O'Connor, González, 2016]. The reasons given for weak adoption is the generality of software architecture and lack of guidance in the specifications of software development methodologies towards software architecture.

8. DESIGN OF SOFTWARE ARCHITECTURE

The macro-architecture design process, or software architecture design process, is a **sub-process** of a software development process, and is typically both **iterative and incremental**.

8.1. PRINCIPLES OF SOFTWARE ARCHITECTURE DESIGN PROCESS

There are a certain principles that any software architecture design process, including the following [Rozanski, Woods, 2005, Chapter 7]:

- **Relevancy.** It must be **driven by stakeholder concerns**, although they are not only inputs to the process, **must balance these concerns effectively** where they conflict or have incompatible implications.

- **Neutrality.** It must be **technology-neutral**, that is, it must **not mandate that the solution** is based upon any specific technology, architectural pattern, or development style, nor should it dictate any particular modeling, diagramming, or documentation style.

- **Integrability.** It must **integrate** with the chosen **software development life cycle**.

8.2. MODELS FOR SOFTWARE ARCHITECTURE DESIGN PROCESS

There are a number of different **philosophies** (or schools of thought) of software architecture design process.

In the following, a number software architecture design process models are presented **chronologically**.

8.2.1. SOFTWARE ARCHITECTURE DESIGN PROCESS: MODEL 1

Figure 10 shows a software architecture design process model given in [Bass, Clements, Kazman, 2003].

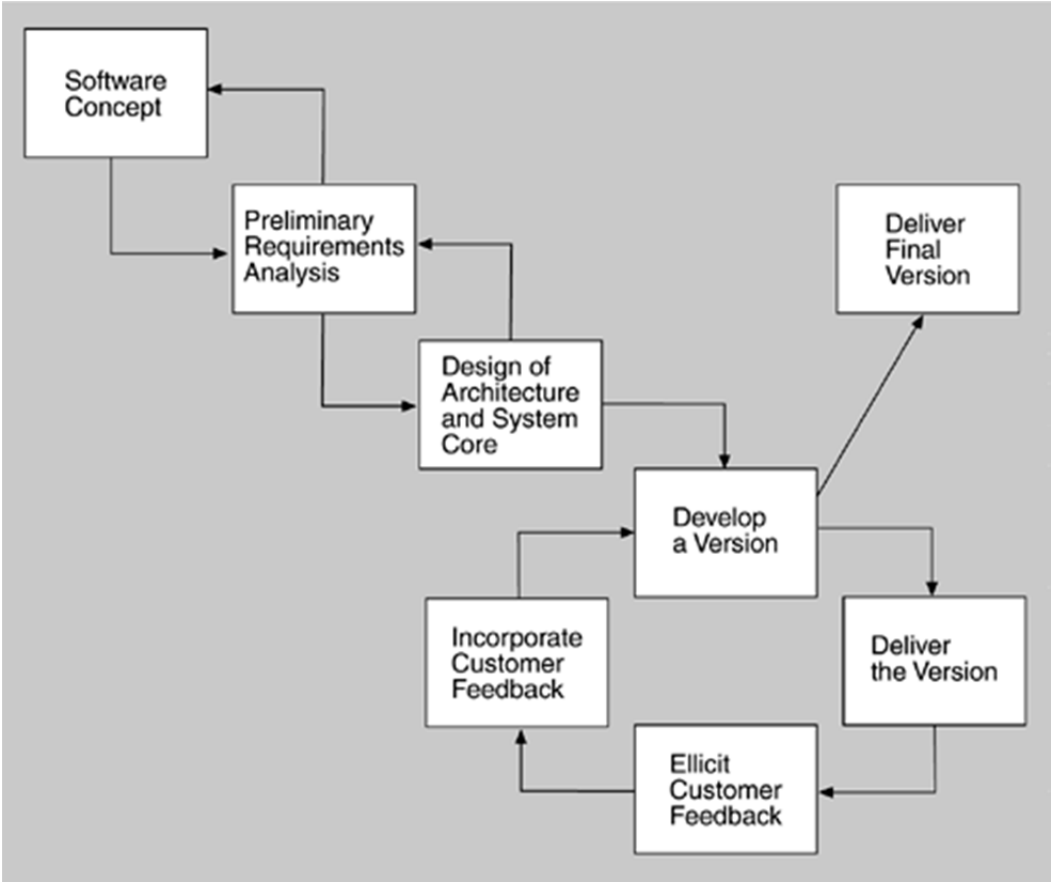


Figure 10. The evolutionary delivery life cycle model. (Source: [Bass, Clements, Kazman, 2003].)

8.2.2. SOFTWARE ARCHITECTURE DESIGN PROCESS: MODEL 2

Figure 11 shows a software architecture design process model given [Rozanski, Woods, 2005, Chapter 7]. It is a bit unusual that there is **no support for evaluation** in the model.

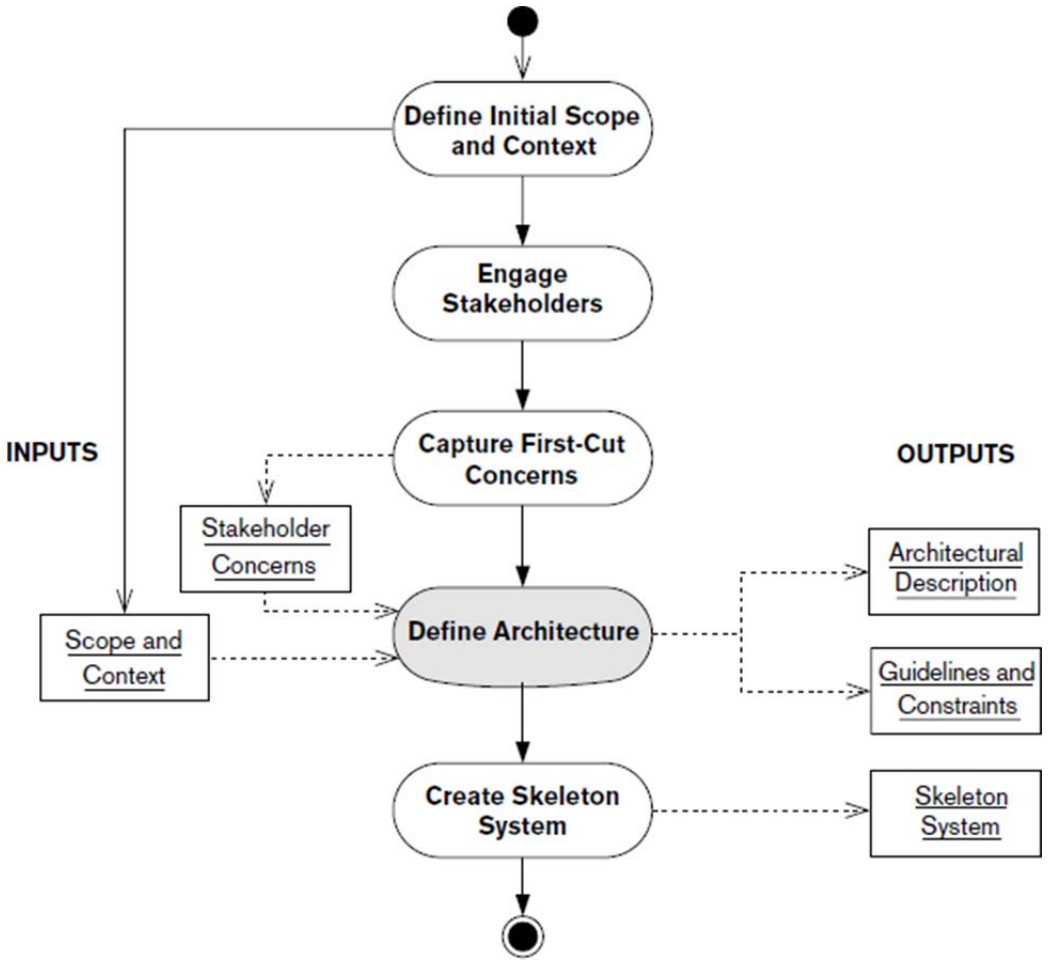


Figure 11. A software architecture design process model. (Source: [Rozanski, Woods, 2005].)

8.2.3. SOFTWARE ARCHITECTURE DESIGN PROCESS: MODEL 3

Figure 12 shows a software architecture design process model given in [Falessi, Babar, Cantone, Kruchten, 2010].

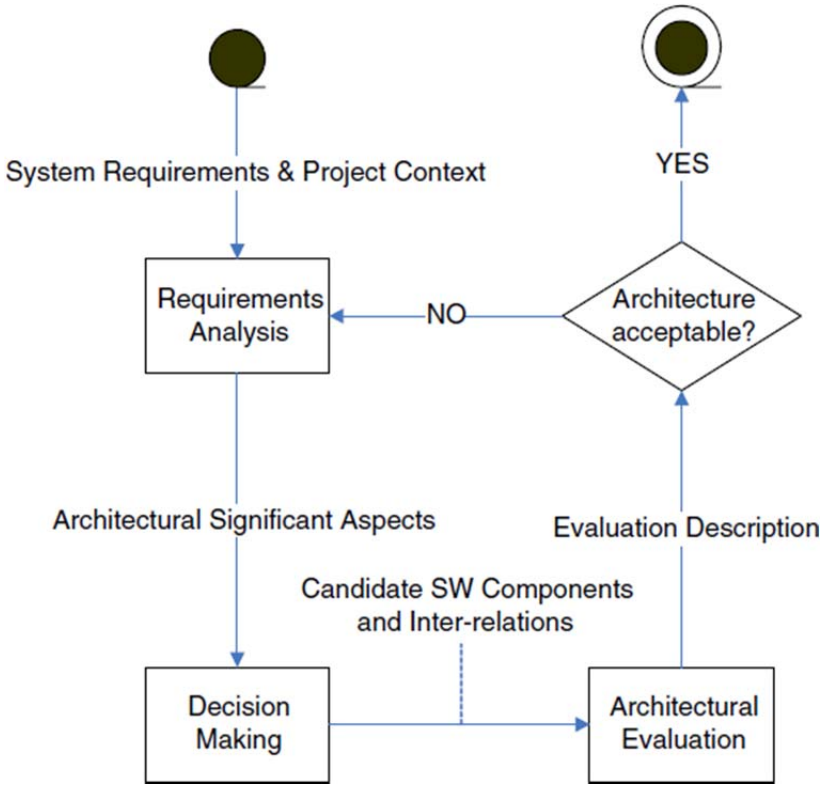


Figure 12. A software architecture design process model. (Source: [Falessi, Babar, Cantone, Kruchten, 2010].)

The stages in this model are:

- **Requirements Analysis.** This involves focusing on stakeholder needs, and identifying requirements that are **architecturally significant**.

- **Decision Making.** This involves making decisions in order to fulfill the stakeholders' needs (as defined in the previous phase), by **exploring** alternatives, and subsequently **selecting** the most appropriate architectural design option(s) from the available alternatives.

- **Architectural Evaluation.** This involves deciding whether and to what degree the chosen alternative solves the problem.

8.2.4. SOFTWARE ARCHITECTURE DESIGN PROCESS: MODEL 4

Figure 13 shows a software architecture design process model given in [Gorton, 2011, Chapter 7].

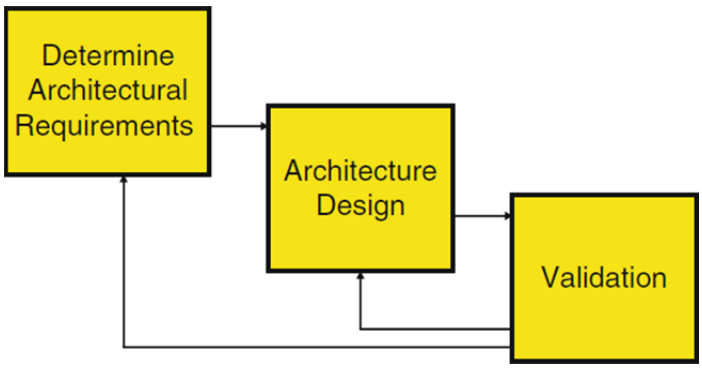


Figure 13. A three-step iterative software architecture process model. (Source: [Gorton, 2011].)

The stages in this model are:

- **Define Architecture Requirements.** This involves creating a statement or model of the requirements that will drive the architecture design.

- **Architecture Design.** This involves defining the structure and responsibilities of the components that will comprise the architecture.

- **Validation.** This involves "testing" the architecture, typically by walking through the design, against existing requirements and any known or possible future requirements.

OBSERVATIONS

OBSERVATION 1

The **inputs and outputs** can **vary** across software architecture design process models.

OBSERVATION 2

The **architecturally significant requirements** are those requirements of a software system that play an important role in determining the architecture of the system. In most models, there is **feedback loop** from **architecturally significant requirements to design**.

The requirements that specify the **quality attributes** of a software system usually are architecturally significant.

The requirements that define the **environment** in which the software system will run are likely to be architecturally significant. (These could include embedded hardware, virtual machines, mobile devices, and so on.) This is because systems running in **different environments** often have **different architectures**.

The requirements of the following type are **not** regarded as architecturally significant[8]:

"The temperature should be displayed in Celsius (not Fahrenheit) on the Web Page."

The above requirement may be crucial to a stakeholder, but its realization has **no impact on the architecture of the software system** (in this case a Web Application or a Web Service). Therefore, it is **not** considered to be architecturally significant.

The requirements of the following type are regarded as architecturally significant:

"The system should provide five nines (99.999%) availability."

OBSERVATION 3

The aforementioned software architecture design process models presented are variations (instantiations and elaborations) of the following:

- **Problem.**
  - Understand.
  - Specify.
- **Solution.**
  - Formulate.
    - Find.
    - Invent.
  - Create Approximation.
  - Evaluate.
    - Automatically.
    - Manually.

[8] URL: http://ulir.ul.ie/handle/10344/3061 .

ACKNOWLEDGEMENT

The inclusion of images from external sources is only for non-commercial educational purposes, and their use is hereby acknowledged.

REFERENCES

[Abrahamsson, Babar, Kruchten, 2010] Agility and Architecture: Can They Coexist? By P. Abrahamsson, M. A. Babar, P. Kruchten. IEEE Software. Volume 27. Issue 2. 2010. Pages 16-22.

[Alexander, 1964] Notes on the Synthesis of Form. By C. Alexander. Harvard University Press. 1964.

[Babar, 2009] An Exploratory Study of Architectural Practices and Challenges in Using Agile Software Development Approaches. By M. A. Babar. The 2009 Joint Working IEEE/IFIP Conference on Software Architecture and 2009 European Conference on Software Architecture (WICSA/ECSA 2009). Cambridge, U.K. September 14-17, 2009.

[Babar, Brown, Mistrik, 2014] Agile Software Architecture: Aligning Agile Processes and Software Architectures. By M. A. Babar, A. W. Brown, I. Mistrik (Editors). Morgan Kaufmann. 2014.

[Bass, Clements, Kazman, 2003] Software Architecture in Practice. By L. Bass, P. Clements, R. Kazman. Second Edition. Addison-Wesley. 2003.

[Booch, 2007] The Irrelevance of Architecture. By G. Booch. IEEE Software. Volume 24. Issue 3. 2007. Pages 10-11.

[Clements, Ivers, Little, Nord, Stafford, 2003] Documenting Software Architectures in an Agile World. By P. Clements, J. Ivers, R. Little, R. Nord, J. Stafford. Technical Report CMU/SEI-2003-TN-023. Software Engineering Institute. Carnegie Mellon University. Pittsburgh, U.S.A. 2003.

[Cockburn, 2008] Using Both Incremental and Iterative Development. By A. Cockburn. CrossTalk. May 2008. Pages 27-30.

[Durdik, 2011] Towards a Process for Architectural Modelling in Agile Software Development. By Z. Durdik. The Seventh International Conference on the Quality of Software Architectures (QoSA 2011) and the Second International Symposium on Architecting Critical Systems (ISARCS 2011). Boulder, U.S.A. June 20-24, 2011.

[Fairbanks, 2010] Just Enough Software Architecture: A Risk-Driven Approach. By G. H. Fairbanks. Marshall and Brainerd. 2010.

[Falessi, Babar, Cantone, Kruchten, 2010] Applying Empirical Software Engineering to Software Architecture: Challenges and Lessons Learned. By D. Falessi, M. A. Babar, G. Cantone, P. Kruchten. Empirical Software Engineering. Volume 15. Number 3. 2010. Pages 250-276.

[Garlan, 2000] Software Architecture: A Roadmap. By D. Garlan. The Twenty Second International Conference on Software Engineering (ICSE 2000). Limerick, Ireland. June 4-11, 2000.

[Gorton, 2011] Essential Software Architecture. By I. Gorton. Springer-Verlag. 2011.

[IEEE, 1990] IEEE Standard 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society. 1990.

[IEEE, 1995] IEEE Standard 730.1-1995. IEEE Guide for Software Quality Assurance Planning. IEEE Computer Society. 1995.

[ISO/IEC, 2008] ISO/IEC 12207:2008. Systems and Software Engineering -- Software Life Cycle Processes. The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC). 2008.

[ISO/IEC/IEEE, 2010] ISO/IEC/IEEE 24765:2010. Systems and Software Engineering -- Vocabulary. The International Organization for Standardization (ISO)/The International Electrotechnical Commission (IEC)/IEEE Computer Society. 2010.

[Jacobson, Booch, Rumbaugh, 1999] The Unified Software Development Process. By I. Jacobson, G. Booch, J. Rumbaugh. Addison-Wesley. 1999.

[Kazman, Kruchten, Nord, Tomayko, 2004] Integrating Software-Architecture-Centric Methods into the Rational Unified Process. By R. Kazman, P. Kruchten, R. L. Nord, J. E. Tomayko. Technical Report CMU/SEI-2004-TR-011 ESC-TR-2004-011. Software Engineering Institute. Carnegie Mellon University. Pittsburgh, U.S.A. 2004.

[Kroll, MacIsaac, 2006] Agility and Discipline Made Easy: Practices from OpenUP and RUP. By P. Kroll, B. MacIsaac. Addison-Wesley. 2006.

[Kruchten, 2004] The Rational Unified Process: An Introduction. By P. Kruchten. Third Edition. Addison-Wesley. 2004.

[Kruchten, Obbink, Stafford, 2006] The Past, Present, and Future for Software Architecture. By P. Kruchten, H. Obbink, J. Stafford. IEEE Software. Volume 23. Issue 2. 2006. Pages 22-30.

[Nord, Tomayko, 2006] Software Architecture-Centric Methods and Agile Development. By R. L. Nord, J. E. Tomayko. IEEE Software. Volume 23. Issue 2. 2006. Pages 22-30.

[Palmquist, Lapham, Miller, Chick, Ozkaya, 2013] Parallel Worlds: Agile and Waterfall Differences and Similarities. By M. S. Palmquist, M. A. Lapham, S. Miller, T. Chick, I. Ozkaya. Software Engineering Institute. Technical Note CMU/SEI-2013-TN-021. Carnegie Mellon University. Pittsburgh, U.S.A. 2013.

[Reyes-Delgado, Mora, Duran-Limon, Rodríguez-Martínez, O'Connor, González, 2016] The Strengths and Weaknesses of Software Architecture Design in the RUP, MSF, MBASE and RUP-SOA Methodologies: A Conceptual Review. By P. Y. Reyes-Delgado, M. Mora, H. A. Duran-Limon, L. C. Rodríguez-Martínez, R. V. O'Connor, R. M. González. Computer Standards and Interfaces. Volume 47. 2016. Pages 24-41.

[Royce, 1970] Managing the Development of Large Software Systems. By W. Royce. IEEE WESCON Conference. Volume 26. August 1-9, 1970.

[Rozanski, Woods, 2005] Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. By N. Rozanski, E. Woods. Addison-Wesley. 2005.

[Sangwan, 2015] Software and Systems Architecture in Action. By R. S. Sangwan. CRC Press. 2015.

[Stephens, 2015] Beginning Software Engineering. By R. Stephens. John Wiley and Sons. 2015.