ISSN NO: 2770-0003
Date of Publication:18-09-2021

General Issues of Applications Architecture Domain Design

Andrii Kopp, Dmytro Orlovskyi, Dorukhan Ersoyleyen
Department of Software Engineering and Management Information Technologies,
National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine
kopp93@gmail.com, orlovskyi.dm@gmail.com, ersoyleyen_1@hotmail.com

Abstract: Applications architecture domain as a part of enterprise architecture is the baseline of any organizational activity, which main goal is to provide the executional environment for business processes in order to deliver products or services to satisfy customer needs and generate revenue. Nowadays, software engineering projects always begin with the architecture design phase, despite the waterfall or agile methodology is used by a software development team. Applications architecture design is the most important and, at the same time, the most error-prone stage of the whole software engineering project. It is well-known, that design shortcomings made on the applications design phase may increase drastically to testing and maintenance phases. Further costs to defects fixing may be hundred times higher in the later project stages in compare to the design stage on which applications architecture is defined.

Keywords: Applications Architecture, Software Development Lifecycle, Software Engineering, Software Design.

INTRODUCTION

Various software development lifecycle (SDLC) models have been introduced within the software engineering and information systems design domain. The simplified taxonomy of SDLC models include [1]:

– waterfall development;
– iterative (incremental) development;
– spiral development;
– agile development.

The traditional waterfall model could be represented as the sequential design process in which progress is seen as steadily descending (like a waterfall) through the stages of [1], [2]:

– requirements gathering analysis – all possible requirements for the system being developed are recorded at this stage and documented in a document with a requirements specification;

– system design – at this stage, the specifications of the requirements of the first stage are studied and a system design is prepared; the system design helps in defining the hardware and system requirements and also helps in defining the overall system architecture;

– system implementation – taking into account the initial data obtained during the design of the system, the system is first developed in the form of small programs called modules, which are integrated in the next stage; then each module is developed and tested for its functionality, which is called unit testing;

– system integration testing – all modules were developed during the implementation phase are integrated into the system after testing each module; after integration, the entire software system is checked for faults and failures;

– system deployment – after completion of functional and non-functional testing; the product is implemented in the customer's environment or released to the market;

– system maintenance – in case if there are some issues in the customer environment patches are being released to fix these problems; also, several improved versions may have been released to improve the product; in general, maintenance is performed to bring some changes to the customer's environment in order to fix some defects appeared or to bring new functionality into the software system.

ISSN NO: 2770-0003
Date of Publication:18-09-2021

Major activities of the waterfall SDLC model [2] are demonstrated in figure 1.

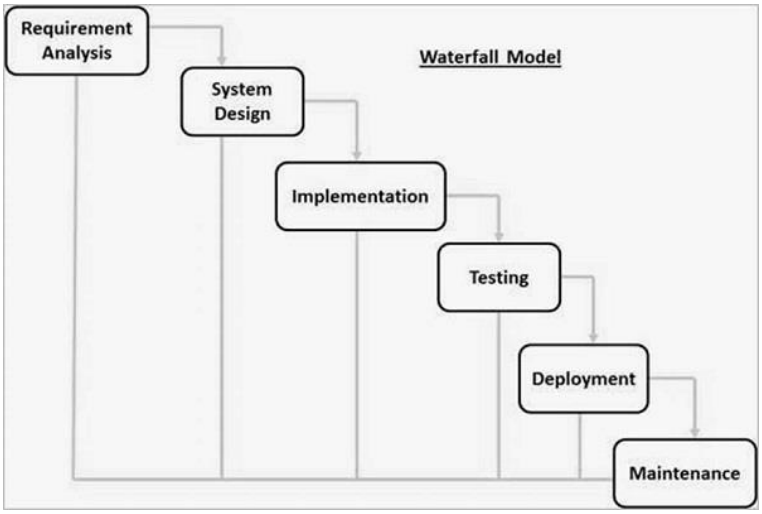

Figure 1. Baseline activities of the waterfall SDLC model [2].

A waterfall model is only meant to transition to a phase when its previous phase is reviewed and tested. Typically, the waterfall model focuses on properly documenting artifacts in lifecycle activities [1].

It is interesting that incremental (also referred as iterative model) or spiral software development models are nothing more that variations of the considered phases but with the retrospective, i.e. software development teams returns back to initial steps one or more times through the iteration to elicit business requirements and make necessary changes to the developed software product. This similarity between the incremental and spiral models in compare to the waterfall software development lifecycle from the presence of basic stages point of view are demonstrated in figure 2. The focus in figure 2 is made on the system design phase, which is essential for the success of software development project.

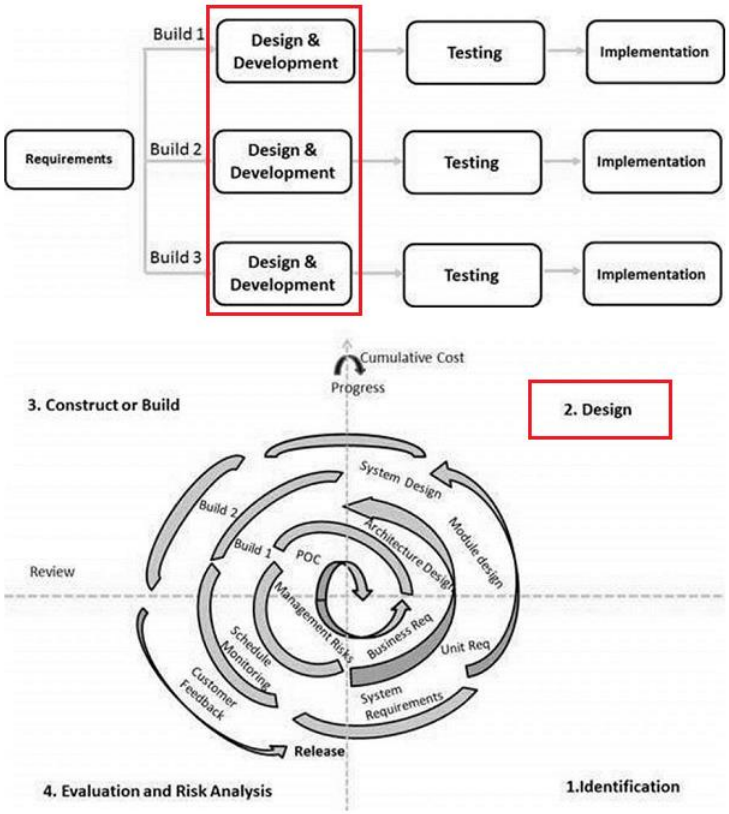

Figure 2. Generic phases of incremental and spiral SDLC models [3], [4].

ISSN NO: 2770-0003
Date of Publication:18-09-2021

Supporters of the agile software development paradigm argue that for any non-trivial project, it is almost impossible to complete the phase of the software product lifecycle before moving on to the next phases. A related argument is that customers may not know exactly what the final software product they need and therefore the software under development need to constantly change [1].

Agile SDLC model brings a lot of flexibility to the software engineering projects, it makes developers and other project team members constantly and directly interact with the customer or customer's representatives (e.g. product owners, business users, or other stakeholders). During their interactions, software requirements constantly elicited and changes are made to the software system being developed. System design stage in such conditions seems extremely vital, since all the frequent changes must be tracked and analyzed for their solidity. The place of system design phase in the agile SDLC model is demonstrated in figure 3.

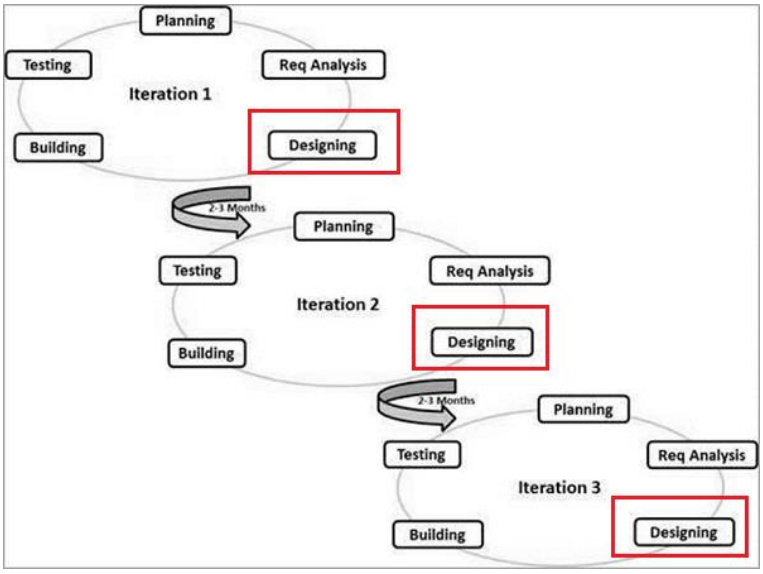

Figure 3. Agile software development methodology SDLC [5].

As it was shown on the Figures 1 – 3, system design phase is always initial despite the SDLC model in use. It does not matter if project has one extremely large "iteration" within the waterfall model, or if project has bunch of iterations with respect to iterative, spiral, or agile software development lifecycles. System design stage is not only the most important phase, but also the most vulnerable phase, since poor design decisions will lead to increasing of costs required to fix such decisions at the future stages of software development project (i.e. at the stages of building, testing, deployment, and maintenance).

MATERIALS AND METHODS

According to research results and practitioners experience, it is important to start testing as early as possible and anticipate possible errors that the developer can make. It is because of the early defects are found, the less will be cost (efforts, man-hours, monetary loses and other resources) of fixing these defects [6]. The diagram that demonstrates such tremendous growth of defect fix costs is demonstrated in Figure 4.

ISSN NO: 2770-0003
Date of Publication:18-09-2021

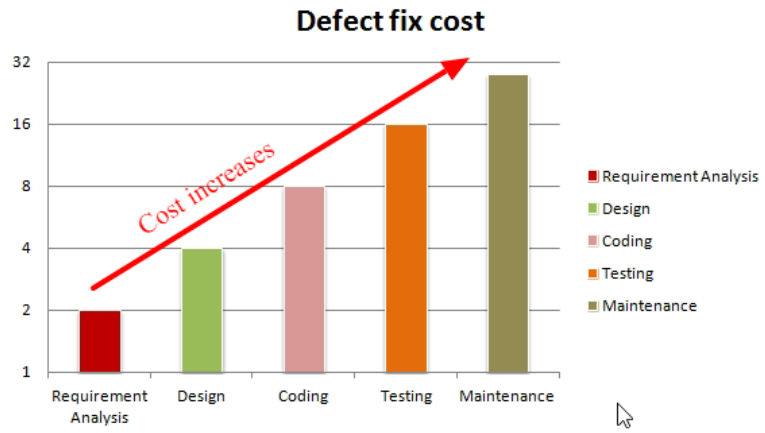

Figure 4. Growth of defect fix cost along the basic SDLC phases [6].

As it is shown in the Figure 4 above, such growth is reality for any kind of SDLC model despite it is agile or linear. Despite in [6] it is stated that testing must be started earlier in order to prevent defects, we assume that systems design should be tested first of all for the possible defects that may appear at the source code development stage.

Therefore, advanced techniques of system design should be used. Here application architecture modeling technique is usually applied to describe the general overview of the system (e.g. to define system's architecture of apply several best practices for systems design) and then move to detailed application components.

It is generally accepted that a well-defined software development process will support the development of quality software products with substantially fewer defects. Some popular examples of process improvement models include the Software Engineering Institute's Capability Maturity Model Integration (CMMI), ISO/IEC 12207, and SPICE (Software Process Improvement and Capability Determination) [1].

Software design patterns are common solutions to solving software systems design problems. The quality of the software can be maintained by reusing design patterns that have been validated in the past. Associated with design patterns [7] is the concept of anti-patterns, which are a common response to a recurring problem that is usually ineffective and counterproductive. Usually, code defects are associated with certain structures in the design, which indicates a violation of fundamental design principles and also negatively affects the quality of the software system design [1].

Specific enterprise application patterns are described by Martin Fowler in [8]:

– domain logic patterns;
– data source architectural patterns;
– object-relational behavioral, structural, and metadata mapping patterns;
– web presentation patterns;
– distribution patterns;
– concurrency patterns;
– session state patterns;
– baseline patterns.

Obviously, all of the considered and basically any existing applications architecture patterns have their advantages and drawbacks [7], which must be taken into account when choosing patterns at the system design phase of SDLC.

RESULTS AND DISCUSSION

Applications architecture models describe software architecture as the system of interrelated application components. Therefore, we can consider the following system design patterns [9], [10], which were described using the ArchiMate modeling language and the applications architecture domain:

ISSN NO: 2770-0003
Date of Publication:18-09-2021

– sequential system design structure (Figure 5);

Figure 5. Sequential system design structure.

– ring system design structure (Figure 6);

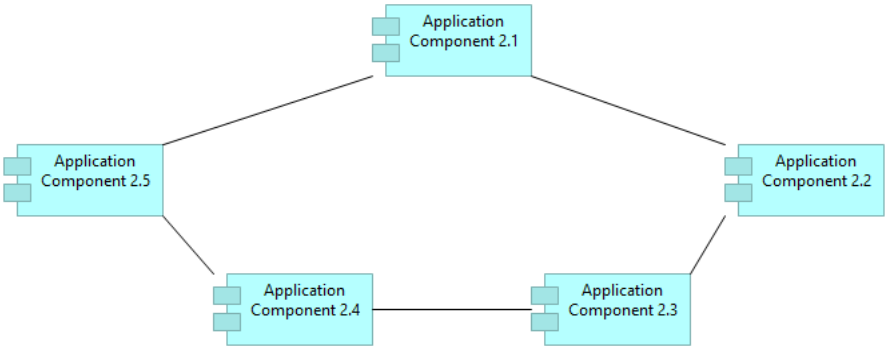

Figure 6. Ring system design structure.

– radial system design structure (Figure 7);

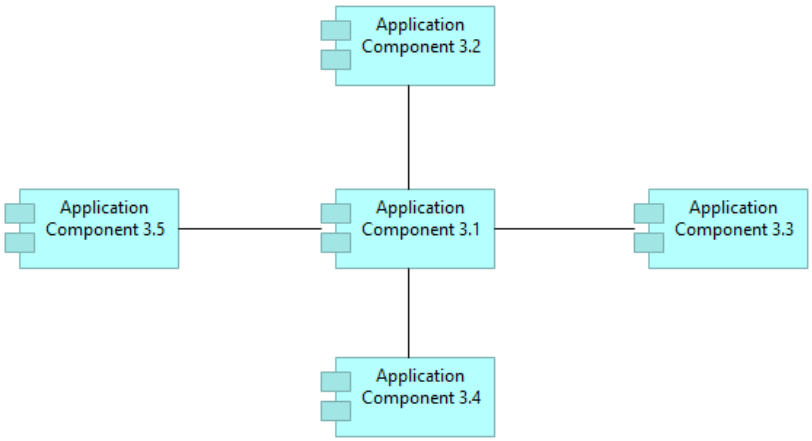

Figure 7. Radial system design structure.

– tree system design structure (Figure 8);

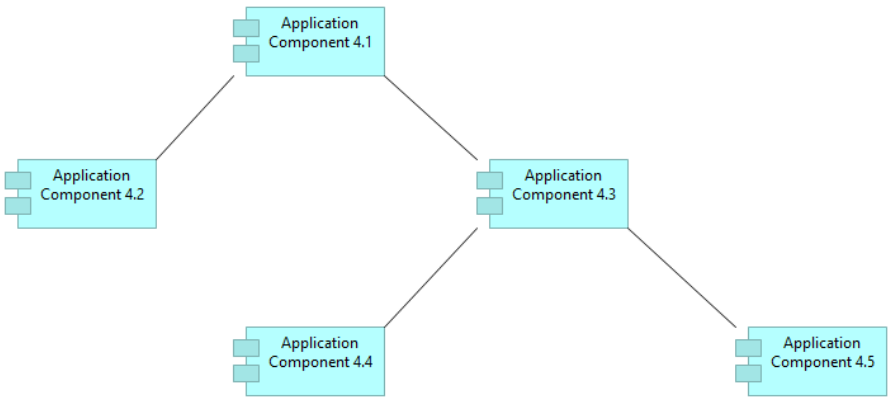

Figure 8. Tree system design structure.

ISSN NO: 2770-0003
Date of Publication:18-09-2021

– mesh system design structure (Figure 9);

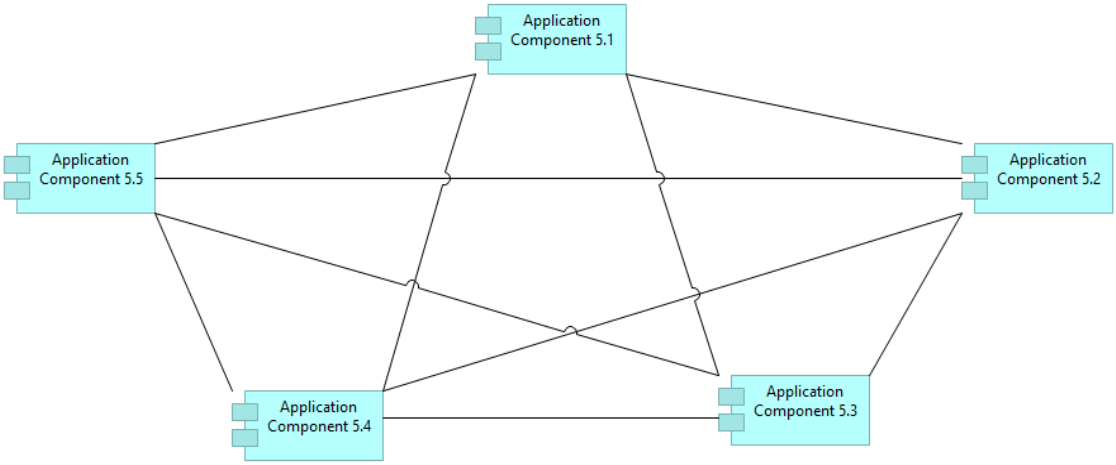

Figure 9. Mesh system design structure.

Based on demonstrated above system design patterns (Figures 5 – 9) there were formulated following inferences:

1. If applications architecture model is the most similar to the sequential pattern, then following recommendation should be obtained “Development: easy, Cost: inexpensive, Flexible: yes, Reliability: moderate, Extension: easy, Robust: no” [29].
2. If applications architecture model is the most similar to the ring pattern, then following recommendation should be obtained “Development: difficult, Cost: moderate, Flexible: no, Reliability: high, Extension: easy, Robust: no” [29].
3. If applications architecture model is the most similar to the radial pattern, then following recommendation should be obtained “Development: easy, Cost: expensive, Flexible: yes, Reliability: high, Extension: easy, Robust: yes” [29].
4. If applications architecture model is the most similar to the tree pattern, then following recommendation should be obtained “Development: easy, Cost: moderate, Flexible: yes, Reliability: high, Extension: easy, Robust: no” [29].
5. If applications architecture model is the most similar to the mesh pattern, then following recommendation should be obtained “Development: difficult, Cost: expensive, Flexible: no, Reliability: moderate, Extension: difficult, Robust: yes” [29].

CONCLUSION AND FUTURE WOK

In this paper was considered a relevant problem of applications architecture design and future models analysis. Its relevance is defined by those fact that designed blueprints of software systems should be carefully checked for all presumable inefficiencies in order to avoid extra efforts and related costs for defects fixing in the later project stages.

ArchiMate language of enterprise architecture modeling is currently considered as the standard representation of applications architecture models that should be analyzed. Such analysis could be achieved by comparing applications architecture models toward the system design patterns also represented using the ArchiMate modeling language [11].

In order to analyze applications architecture models, the following tasks should be considered as part of future research:

1. Propose an approach to applications architecture models analysis.
2. Discover an applications architecture models analysis problem domain with the help of structural, functional, workflow, and data flow analysis.
3. Elicit requirements to the software solution for applications architecture models analysis, including business rules, use cases, and quality constraints.
4. Design and develop the software solution, and use it for applications architecture models analysis.

ISSN NO: 2770-0003
Date of Publication:18-09-2021

ACKNOWLEDGEMENT

This study was prepared at the Department of Software Engineering and Management Information Technologies of National Technical University "Kharkiv Polytechnic Institute" by Andrii Kopp (Associate Professor, Ph.D), Dmytro Orlovskyi (Associate Professor, Ph.D., Docent), and Dorukhan Ersoyleyen (B.Sc. Student).

REFERENCES

1. Tekinerdogan B. et al. Quality concerns in large-scale and complex software-intensive systems. Software Quality Assurance: In Large Scale and Complex Software-Intensive Systems, 2016. 1-17 pp.
2. Waterfall Model, https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm, last accessed: 20.11.2020.
3. Iterative Model, https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm, last accessed: 20.11.2020.
4. Spiral Model, https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm, last accessed: 21.11.2020.
5. Agile Model, https://www.tutorialspoint.com/sdlc/sdlc_agile_model.htm, last accessed: 21.11.2020.
6. Principles of Software Testing, https://medium.com/@er.shankar.acharya/7-principles-of-software-testing-2d1ce74a6440, last accessed: 24.11.2020.
7. 10 Common Software Architectural Patterns in a Nutshell, https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013, last accessed: 15.12.2020.
8. Catalog of Patterns of Enterprise Application Architecture, https://martinfowler.com/eaaCatalog/, last accessed: 20.01.2021.
9. Shyrokov L.A. Control systems research. MSIU, 2010. 167 p.
10. Bisht N., Singh S. Analytical study of different network topologies. International Research Journal of Engineering and Technology, 2015. Vol. 1, No 2. 88-90 pp.
11. Kopp A. M., Orlovskyi D. L., Ersoyleyen D. Applications architecture analysis based on design patterns and image recognition. In: Information technologies: science, engineering, technology, education, health. MicroCAD-2021. Kharkiv, Planet-Print, 2021. 14 p.