Queueing Modeling in the Course in Software Architecture Design

Vira Liubchenko
System Software Department
Odessa National Polytechnic University
Odessa, Ukraine
lvv@opu.ua

***Abstract*—The studying of dependencies between architecture design decision and the quality attribute is difficult for students because of the high level of abstractness. The article introduces an idea to use the queueing modeling for examining the software efficiency of the different decisions. Because of study purposes, the only requirement for the model is a similarity to architecture topology; the model parameters are chosen to provide comparability of analyzed decisions. In the article, there are discussed the result of involvement the queueing modeling into the course in Software Architecture Design and defined the directions of future work.**

Keywords— software design; architecture style; queueing modeling.

I. INTRODUCTION

Ukraine owns the fastest-growing number of IT professionals in Europe; it increases by 20% per year. That has been possible due to a new generation of young developers and engineers – during the last four years the number of IT specialists has increased from 42.4K to 91.7K IT specialists. By 2020, the number of software developers, engineers, and other IT specialists in Ukraine could attain to 250K people [1].

Since the industry is rapidly growing, the issue of education has occupied a significant place. The business requires not only modern knowledge but also appropriate comprehension, skills and mindset from graduates. Sometimes for the universities, it is complicated to meet such requirements because of external causes. For example, software engineering is an empirical field of study, based on the experience of engineers. However, some key areas of software engineering provide limited support for teaching purposes. Therefore, teachers need to use the models in the study process to demonstrate the high-level abstractions and dependencies.

The purpose of this paper is an exploration of queueing modeling for examining the software quality characteristics in the course on Software Architecture Design.

The rest of the paper is organized as follows. Section 2 briefly describes the challenges for the course from the modern tendency in software architecture design. Section 3 presents the example how to use queueing modeling for analyzing the efficiency of architectural decisions. Section 4 discusses some results of the in-class implementation of the proposed idea.

II. RECENT TRENDS AND TEACHING CHALLENGE IN THE COURSE IN SOFTWARE ARCHITECTURE DESIGN

Software engineering is an engineering discipline that is concerned with all aspects of software production [2]. Software Engineering Body of Knowledge defines 15 key areas; one of them is Software Design. The result of software design describes the software architecture – that is, how software is decomposed and organized into components – and the interfaces between those components [3]. It represents fundamental decision profoundly affected the software and the development process. The theory of high-level (architecture) design has been developed from the mid-1990s, this gave rise to some exciting concepts about architectural design – architectural styles and architectural tactics. An architectural style provides the software's high-level organization; an architectural tactic supports particular quality attributes of the software. Both – architectural styles and architectural tactics – are kinds of design patterns documented a recurring problem-solution pairing within a given context. A design pattern is more than either just the problem or the solution structure: it includes both the problem and the solution, along with the rationale that binds them together.

The historically first architectural pattern is stand-alone systems, in which the user interface, application 'business' processing, and persistent data resided in one computer. Such architectural styles as layers, pipe-and-filters and model-view-controller were born from this kind of systems. The explosive growth of the Internet cause that most computer software today runs in distributed systems, in which the interactive presentation, application business processing, and data resources reside on loosely coupled computing nodes and service tiers connected by networks. The most known architectural styles for distributed systems are client-server, three-tiers and broker [4].

A recent trend in the area of architectural design is service-oriented architecture and microservices. Instead of building a single monolithic or distributed application, the idea is to split the application into a set of smaller, interconnected services,

which typically implement a set of distinct features or functionality. Each microservice is a mini-application that has its hexagonal architecture consisting of business logic along with various adapters. The microservices architecture style features some significant benefits: it tackles the problem of complexity, enables each service to be developed and deployed independently, and enables each service to be scaled independently. Therefore, they are not the silver bullet. Like every other technology, the microservices architecture style suffers drawbacks: small service size, the complexity of a distributed system, the partitioned database architecture, the complexity of testing and deploying.

Instead of recent trend software architect ought to analyze different aspects to take a decision, and often the "old-fashion" architecture styles are more appropriate for the developed application. Usually, the impact on quality attributes and tradeoffs among competing for quality attributes are the basis for design decisions. Professional software architects, who have risen from senior software developer, feel architecture drivers at the intuition level. However, what about the students?

Curriculum Guidelines [5] insist on teaching course in the architecture design of complete software systems, building on components and patterns for an undergraduate student. The core model for architecture description is 4+1 view model designed for describing the architecture of software-intensive systems, based on the use of multiple, concurrent views [6]. The 4+1 model illustrates the high-level design of the system from different points of view, but it does not provide the possibility of modeling for system dynamic. A primary lesson of the course for students is to learn how to provide quality (or non-functional) attributes in their designs, so they should realize how architectural decisions impact on the quality attributes. It may be the most difficult subjects to teach to students lacking in considerable system experience. Therefore, the teacher needs visual and straightforward modeling tools to analyze quality scenarios for different architectural decision.

One of the appropriate tools is queueing modeling. This kind of models is clear and visual. The only problem concerned the queueing model in the framework of Software Engineering curriculum was the lack of students' knowledge. Therefore, we should have remarkably simplified the model definition.

III. The Examples of Queueing Modeling for Architecture Analysis

Queueing networks are a powerful abstraction for modeling systems involving contention for resources; they are especially useful in modeling computer communication systems. In this model, the computer system is represented as a network of queues which is evaluated analytically. A network of queues is a collection of service centers, which represent system resources, and customers, which represent users or transactions. The queueing networks are easy to use for students due to many software solutions for queueing modeling. The queueing modeling is usually used when deciding the needed resources and the core problem is the definition of adequate parameters.

Naturally examining how the architecture style impact on software efficiency does not need a very accurate model. It is enough to provide the possibility of result comparability. Therefore, the task of queueing modeling becomes simpler than in usual operation research case. Let us demonstrate the ideas with two examples.

A. Splitting the Monolith

Suppose the software efficiency is examined quality attribute, and the students have to consider how the different levels of software "granularity" influence the efficiency. For the purposes, students compare three architectural designs: monolith – a typical enterprise application consists of three layers (presentation, business logic and data-access), split frontend and backend, extracted services (see Fig. 1).

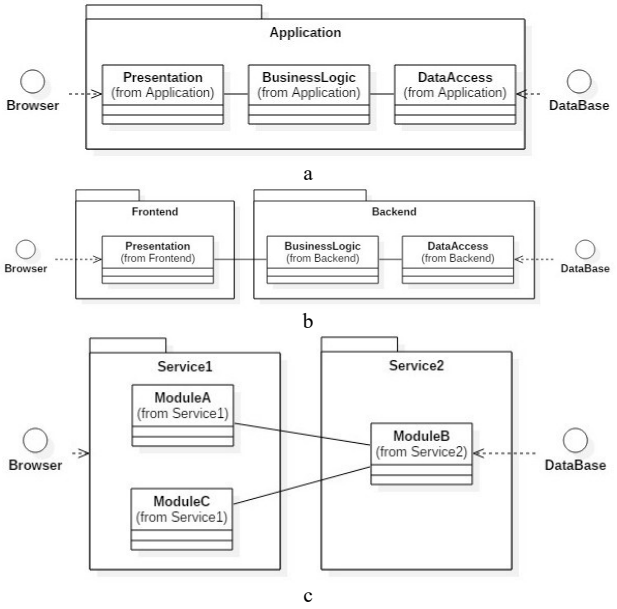

Fig. 1. Architectural design: a – monolith; b – split frontend and backend; c – extracted services.

Remember, queueing modeling is the demonstration tool only. Having regard to the purpose, we can choose the parameters as simple as possible to focus students' attention on software efficiency. Let us suppose all queues are M/M/1 queue. As the controlled attribute of software quality is efficiency, students are recommended to analyze the average time in the system as productivity metric and average utilization as resources utilization metric.

The first case is queueing model for monolith architecture (see Fig. 2). There are two types of clients, suppose the average time interval between two arrivals from mobile clients is 4 time units (for example, seconds); the average time interval between two arrivals from web clients is 6 time units. Both clients – mobile and web – retrieve the data by making a single REST call to the application, suppose each remote transaction takes 1 time unit. The Gateway serves as the sole entry point into the Application (like Façade pattern). Suppose each application action takes 2 time units.

In this case, the average time in the system is 0.32, and the average application utilized is 0.99. Because of high utilization,

the requests are waiting for processing; the average size of the input buffer is 28.5 so that additional component for a buffer of requests should be provided in design.

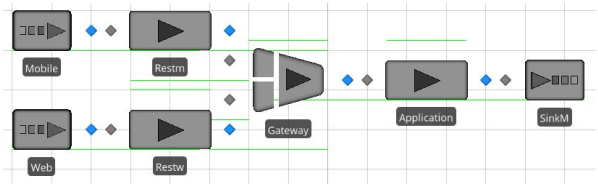

Fig. 2. Queueing model for monolith architecture

The second case is the queueing model for split frontend and backend (see Fig. 3). For comparability of results, both average time intervals for clients remained the same as well as the time for remote transaction and actions of software. The only difference is the topology of components.

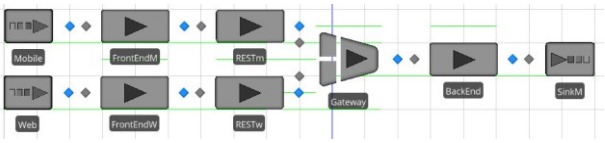

Fig. 3. Queueing model for split frontend and backend

Such design decision increases the average time in the system to 0.91. Instead of this utilization of the software components decreases to 0.48 and 0.63 for frontend and backend. The only backend needs the buffer; the average size of the input buffer is 1.2.

The third case is the queueing model for extracted services (see Fig. 4). As in the previous case both average time intervals for clients, the average time for remote transactions and actions were not changed. Additional remote transaction appeared because of modeling the connection between separated services.

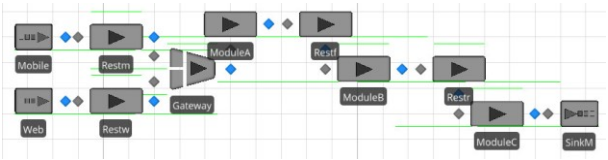

Fig. 4. Queueing model for extracted services

From the productivity point of view, this refactoring is better than the previous one, the average time in the system is 0.61, but from the resources utilization point of view is worse, the average utilization of all services is 0.33. Also, there is no need for providing additional buffers.

B. Schema of Batch Processing

The queueing modeling support students not only in the comparison of structural solutions but also in the study of the implementation of particular components of the software system. For example, when students are studying Big Data architecture style, they should examine the impact of batch processing schema. Batch processing is a component processed data files using long-running batch jobs to filter, aggregate, and otherwise, prepare the data for analysis.

Suppose the software efficiency is examined quality attribute, and the students have to consider how the different schema of batch processing influence the productivity. For the purposes, students compare two approaches: MapReduce programming model and user-defined functions of the data warehouse.

MapReduce model is a specialization of the split-apply-combine strategy for data analysis and consist of three operations. Operation Map computes a given function for each data item and emits value of the function as a key and item itself as a value. Operation Shuffle groups the data items based on the output keys. Operation Reducer obtains all items grouped by function value and processes or saves them. The queueing model for MapReduce model is shown in Fig. 5.

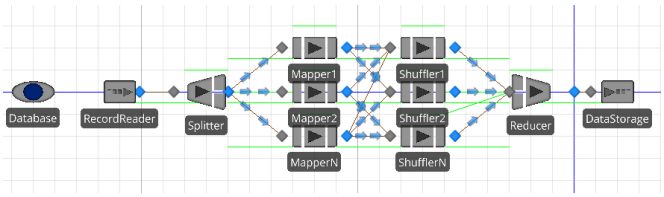

Fig. 5. Queueing model for MapReduce model

A user-defined function (UDF) implements in program language (e.g., Java) the program logic, which is difficult to simulate in request language (e.g., HiveQL). UDF realizes the logic of data sample processing with all required calculations. Such solution transfers the implementation of particular costly operations on data samples in the data warehouse, which reduces the program execution time, eliminates the necessity of repeatedly data extraction from the repository and recording back, separates the functional duties of the data warehouse and the custom program. The queueing model for UDF solution is shown in Fig. 6.

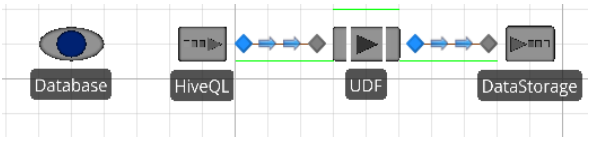

Fig. 6. Queueing model for UDF solution

In the experiment framework, the students compare the average time in the components built under each of both schemas with different data volumes. Accordingly with the results of simulation UDF-solution in faster than MapReduce-based one, but the gap between two solution decreases with increasing of data volume. The students face the complex trade-off situation for software architect: the implementation of UDF is complicated; the benefit from UDF depends on the data volume, to take decision, abstract “big data analysis” is not enough. Such situation is quite difficult for text illustration; the queueing modeling gives the possibility to realize “what-if” experiment.

IV. Analysis of Result

Course in Software Architecture Design is taught in the framework of Software Engineering curriculum for 3rd-year students. Traditionally the course is challenging because of the

high level of abstractness. The timeframe does not provide the possibility for experiments with software prototypes so that students practice thought experiments to examine the dependencies between architectural styles and quality attributes.

Introduction the queueing modeling into the course was quite risky because the students were very unfamiliar with such type of modeling. It causes the necessity for the maximum simplicity of the model. Last year we introduced the queueing modeling for efficiency study (one example of the analyzed scenario is described in Section 3) into the course in Software Architecture Design. Let us discuss the results.

To avoid Hawthorne effect [7], the blind experiment was held; the students were not informed about participating in the research. Two different groups of students were involved in the test. The sampling of the students who participated in the experiment was homogeneous accordingly the distribution of their average score.

The following Table 1 presents a summary of students' result demonstrated in homework assessment and a final exam. We consider only those students, who made the task about efficiency characteristic. In addition, we transform the grades for each task into the universal scale.

TABLE I. THE RESULT OF EFFICIENCY STUDY

Grade	Without queueing modeling		With queueing modeling	
	assessment	*exam*	*assessment*	*exam*
Perfect	2.1 %	0.0 %	6.8 %	4.5 %
Good	60.4 %	47.9 %	70.5 %	63.6 %
Satisfactory	20.8 %	25.0 %	15.9 %	22.7 %
Unsatisfactory	16.7 %	27.1 %	6.8 %	9.1 %
Task success	83.3 %	72.9 %	93.2 %	90.8 %
Quality	62.5 %	47.9 %	77.3 %	68.1 %

Table 1 shows the queueing modeling led to improving the learning outcome quality by 14.8% for assignment and 20.2% for the exam and growing the task success by 9.9% and 17.9% respectively. In addition, the gap in the learning outcomes quality and task success achieved at assignment and exam was reduced, respectively, by 8.0% and 5.4%, so the forgetting effect was reduced.

V. CONCLUSION AND DISCUSSION

The main challenges of Software Engineering curricula are the abstractness and empiricism. The natural tool for teaching are different kinds of the model; the most common of them are the models in UML. Sometimes models in UML remain too abstract for students, and teacher should introduce another means.

The natural model for software architecture design seems to be queueing model. The problem is the students have not studied this class of model. Because of this issue, the parametrization of the queueing model was realized in a primitive way. Instead of inaccuracy of model, it demonstrated the properties of different design decision successfully, and students improved the topic understanding.

The future works ought to be realized in two directions. The first one is introducing the model for other software quality attributes. The second one is developing different analysis scenario based on adequate parametrization schemas.

ACKNOWLEDGMENT

All simulation experiments in the course and the article were realized with Simio Personal Edition software. UML diagrams were drowned with StarUML software.

REFERENCES

[1] Ukrainian IT Market in Numbers and Facts, https://blog.softheme.com/ukrainian-it-market-in-numbers-and-facts/, last accessed 01/07/2018.

[2] I. Sommerville, Software Engineering. London: Pearson, 2010.

[3] Guide to the Software Engineering Body of Knowledge, Version 3.0, P. Bourque and R. E. Fairley, Eds. IEEE Computer Society, 2014.

[4] F. Buschmann, K. Henney, and D. C. Schmidt, Pattern-Oriented Software Architecture. Volume 4: A Pattern Language for Distributed Computing. Hoboken, NJ: Wiley, 2007.

[5] Joint Task Force on Computing Curricula: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Technical Report. New York, NY: ACM, 2015.

[6] P. Kruchten, "Architectural Blueprints – The "4+1" View Model of Software Architecture," IEEE Software, vol. 12(6), pp. 42–50, 1995.

[7] G. D. Gottfredson, "Hawthorne Effect," in Encyclopedia of Statistics in Behavioral Science, B. S. Everitt and D. Howell, Eds. Hoboken, NJ: Wiley, 2005.