Software Architecture Design of Software Developed in Student Engineering Teams

Ihor Polataiko

Department of Information Technology
Vasyl Stefanyk Precarpathian National University
Ivano-Frankivsk, Ukraine

***Abstract*—This paper presents the process of architecture design of the software, intended for development in student engineering teams. Software architecture is a primary driver of software quality. Considering the fact, that students with little experience are working on software, that is intended for real-world usage in the educational institution, it is required to define a deliberate process of software architecture design. The process, developed as the result of the performed research, defines the main stages in architecture design, describes the set of related documentation, and preferable ways of decision-making and validation.**

Keywords—Software Engineering, Software Architecture, Process, Education, Student Teams.

I. INTRODUCTION

To grow successful software engineers, educational institutions, that teach software engineering-related fields should provide students not only with theoretical knowledge and practical assignments but also with a significant amount of production-close development experience. Having an environment, where students with very little or without industry experience could be able to practice and gain required skills is essential. Also, institutions, usually, have some category of students, which are typically in their senior years at the institution, who has some good amount of industry experience (1 or 2 years) and would like to teach and mentor more junior students. This type of experience could be incredibly useful for the mentioned category of students because it gives them the ability to better structure their own knowledge while teaching other students, and receive hands-on experience of technical leading of the projects. Apart from that, most educational institutions have an information technology (IT) department, that manages and develops software for internal needs. Students engineering organization, called PNUdev, was created and currently functions at Vasyl Stefanyk Precarpathian National University.

Software, developed by the teams, that form student software engineering organization should be of high enough quality. During the time of organization functioning, it was observed, that most of the significant reworks to the code changes, made by team members in form of merge requests, were due to design decisions. It was decided to define an upfront architecture process, which teams should follow for each project, that would help to reduce the number of reworks and increase the overall quality of the architectural design of the software along with increasing the level of awareness of the architectural decisions.

II. RELATED WORK

Multiple studies were conducted in the field of innovative approaches to software engineering education and project management processes. Paper [1] describes an application of short-term projects in cross-cultural environments for improving the soft and technical skills of the students. In papers [2], [3] researches related to increasing the level of student motivation and engagement in project-based learning are described. Papers [4], [5], focus on the development of real-world software by students as a vital aspect of engineering education. The research described in the paper [6] development management process, based on reflective weekly monitoring, for the context of

student projects. Finally, studies [7] and [8] are related to using code review as a learning tool for software engineering students. Also, the studies [9], [10] were using parts of the software products, developed by PNUdev, as supportive tools.

III. METHODOLOGY

During the research, both the process of architecture decisions making and the architectural template document were developed as the optimal process and result of the application architecture definition phase in the given context of the student software engineering teams, working on the project. An architectural template helps to unify the form in which the decisions regarding the architecture and design are presented. Decisions are made in the group meetings, where all the team members are included.

The first phase of the architectural definition process is the definition of contexts. At this stage, features are grouped together into contexts. Contexts should have a minimal dependency and relations between each other. If the product is not new, contexts from the architectural template of the previous project are used as a base. In case, new contexts are identified, they should be marked as “new” in the architectural template. Contexts should not share domain data models but can reference aggregate roots from different contexts. For each new context, the decisions regarding dependencies on other contexts, level of splitting, and communication with other contexts should be made. The level of splitting can be either on code level: separate module or component, or deployment level: separate application. Communication can be either synchronous or asynchronous via messaging broker. Asynchronous communication is preferred. As the result of this phase, the diagram of contexts should be created which presents all the contexts, dependencies between them, and ways of communication. The diagram should include deployment boundaries as well. Also, sometimes, one context can be spitted across a couple of deployment units, e.g. back end and front end, which have the same contexts but run in different environments. An example of the most basic context diagram without dependencies between contexts is presented in figure 1.

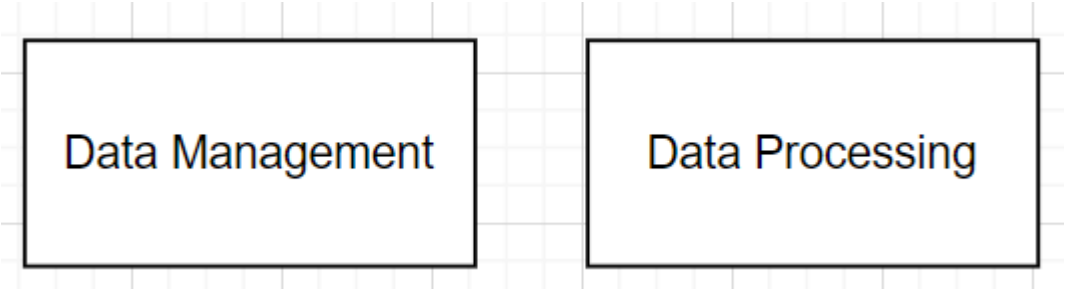

Figure 1. Example of context diagram without dependencies

After contexts are defined, there should be a process to define which of them might be partially or fully covered by the existing open source software systems in order to not reinvent the wheel. In this case, requirements might be updated to focus on integration with that software for the contexts, that are outsourced. Also, at this phase, doubts regarding the technical achievability of the requirements should be cleared out by means of building small proofs of concepts for each of them, which is assigned to some project team members. If too much non-clarity and questions arise during this phase, the project might be suspended and a software research project might be conducted with the team if the context doesn’t imply strict timelines.

The next phase was identified as domain data model definition. For each of the contexts entities have to be identified, based on the requirements for the features. Nouns in the requirements often represent some domain entity. All the functional requirements should be analyzed one by one to identify most of the entity attributes needed (the ones, that were missed, will be added during development). The result of this phase should be a domain data model diagram where all domain entities, their attributes, relations, and fields by which they will be accessed, are represented. If the context existed in the previous project related to the product, then the domain data model diagram

should be the updated version of the diagram from the previous project. Figure 2 presents an example of a domain data model diagram for the context.

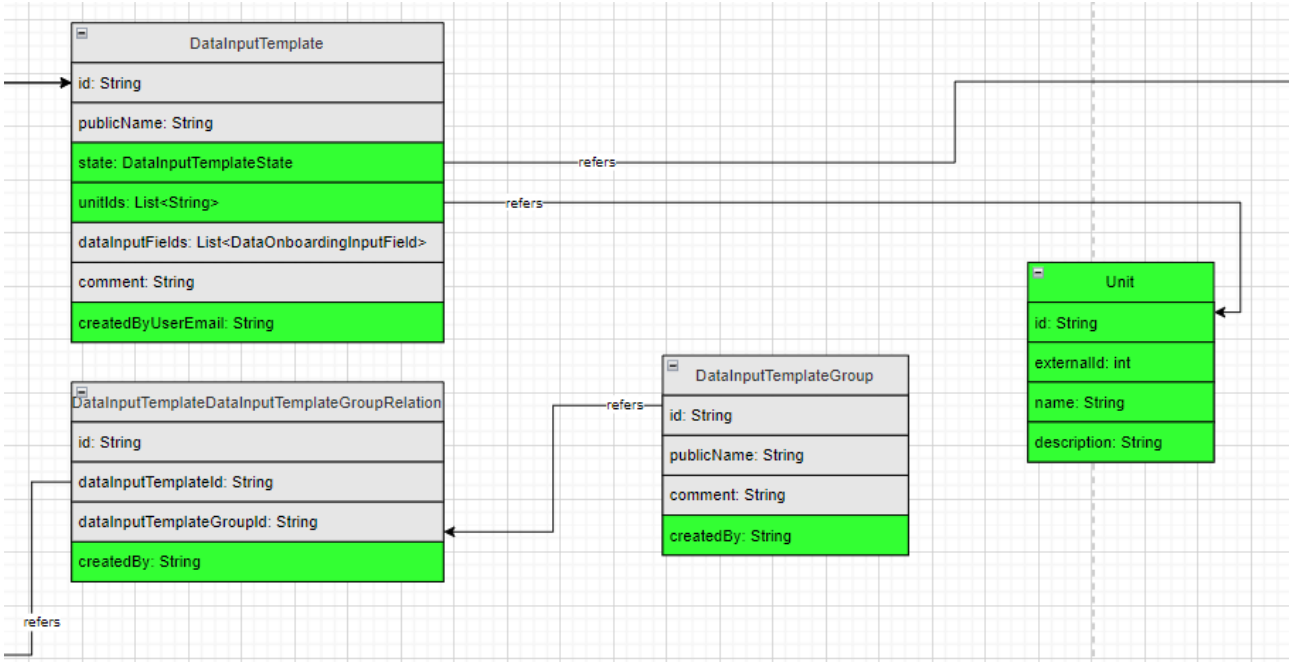

Figure 2. Example of a domain data model diagram

After the contexts and data model are defined, a detailed components diagram should be created. The diagram should specify applications created, show the communication of different components, integration with other systems, and specify the infrastructural components, such as databases, centralized caches, etc. At this phase, non-functional requirements have to be considered, when choosing different technologies and approaches. The decision regarding technical stack, usage of caches, serverless technologies, reactive vs non-reactive drivers, server-side rendering vs single page application should be made at this phase, based on the requirements and team experience. If the product is not new, the component diagram from the previous project should be used as a base. Figure 3 presents an example of a component diagram.

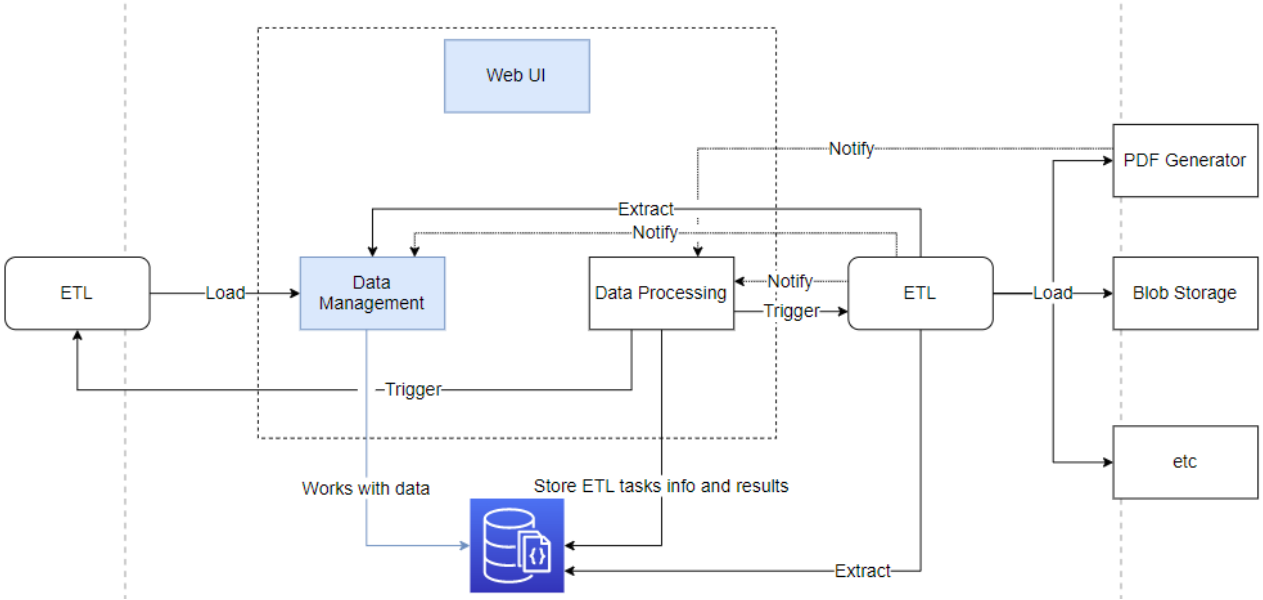

Figure 3. Example of a component diagram

For all parts of the project, which implies exposing the API either outside of the context or outside of the deployment unit, the APIs should be documented and added to the template with reference to the related feature. Apart from that, all the APIs used for integration with other

software systems should be also added to the architecture template.

The resulting architectural template should contain a context diagram, domain data model diagram for each context, component diagram, list of the APIs to be developed, and list of APIs for integration. It should be reviewed and validated carefully at a specific meeting, where all the involved people go through the filled architectural template for the project, discuss the questions and validate the architecture through the mental simulation of execution of functional requirements, having the APIs, components, and data models defined. Based on the results of the review, decisions, and the architectural template might be refined and reviewed again if needed.

IV. Results

The research was performed by means of evaluating the experience of creation and leading of the PNUdev organization, experimentation, and path of trial and error. The organization functions for about 2 years, at the time of writing this paper. During this time, 5 software products were worked on. Some of them continue to evolve in the current projects. Since the start of applying the described template, the number of code review comments, related to the architectural design decisions, is reduced by 90%. The remaining 10% are related to human errors and lower-level design decisions.

V. Discussion

A proposed process of software architecture design allows student engineering teams to have a straight and comprehensive process of defining architectural decisions upfront. Considering, that students, joining the engineering teams, usually, do not have enough expertise to be able to make good quality architectural decisions, the process helps the team to focus on development, during the development phase, and eliminates the possibility of making design decisions of low quality, because the primary aspects of the architecture for the project are defined and reviewed by the whole team and by more experienced colleagues, who manage the teams. Usually, the technical manager of the team, who has some industry experience, drives the meetings dedicated to architectural decisions makings. It is beneficial for both the technical manager and less experienced members of the team. Technical managers practice the skills of architectural decisions making while explaining the reasons for the decisions to the rest of the team, and other members of the team are able to follow the process of architectural decision-making and participate in it.

The model is not intended for application outside of the context of the student software engineering teams. Software engineers of a more senior level are able to make most of the decisions in the scope of the implementation. However, API-driven design, during which the list of APIs is negotiated and defined upfront, that model imposes is a good practice, independently of the context. Also, the described process assumes, that project development is made in medium-term iterations, hence projects, that practice continuous delivery of features to a production environment, would, most certainly, not be able to apply the described process, because it would be a huge overhead.

VI. Future research

The aspects, related to measuring and assuring the quality of architectural decision-making can be a focus of future research on the given topic.

VII. Conclusion

As the result of the performed research and analysis of the experience of software design and development performed by the teams which are part of the PNUdev organization, which functions at Vasyl Stefanyk Precarpathian National University, the process of architecture design and related set of documentation, called architecture template, was developed.

VIII. Acknowledgment

The research has been performed at and supported by the Department of Information Technology of the Vasyl Stefanyk Precarpathian National University.

IX. Disclosures

The author of this paper is a student of the Vasyl Stefanyk Precarpathian National University at the time, of the research and writing this paper.

References

[1] 1. H. Akerlund, G. Audemard, H. Bollaert, V. Hayenne-Cuvillon, A. Hlobaz, M. Kozlenko, P. Milczarski, J.C. Monteiro, J. Morais, D. O'Reilly, P. Possemiers, and Z. Stawska, "Project GGULIVRR: generic game for ubiquitous learning in interactive virtual and real realities," in EDULEARN20 Proceedings of the 12th International Conference on Education and New Learning Technologies, July 6-7, 2020, pp. 5973-5979, doi:10.21125/edulearn.2020.1566.

[2] P. Morais, M. J. Ferreira and B. Veloso, "Improving Student Engagement With Project-Based Learning: A Case Study in Software Engineering," in IEEE Revista Iberoamericana de Tecnologias del Aprendizaje, vol. 16, no. 1, pp. 21-28, Feb. 2021, doi: 10.1109/RITA.2021.3052677.

[3] R. K. Pucher, A. Mense, H. Wahl and F. Schmöllebeck, "Intrinsic motivation of students in project based learning," in Transactions of the South African Institute of Electrical Engineers, vol. 94, no. 3, pp. 6-9, Sept. 2003.

[4] R. J. Machado, P. Guerreiro, E. Johnston, M. Delimar and M. A. Brito, "Work in progress - IEEEXtreme: From a student competition to the promotion of real-world programming education," 2009 39th IEEE Frontiers in Education Conference, 2009, pp. 1-2, doi: 10.1109/FIE.2009.5350540.

[5] Buhari, S. Valloo and H. Hashim, "A Streamlined Approach to Enhance the Capacity of Undergraduate IT Students to Deliver High Quality and Demand-Driven Final Year Project: A Conceptual Framework on Collaboration between Industry and University," 2017 7th World Engineering Education Forum (WEEF), 2017, pp. 910-914, doi: 10.1109/WEEF.2017.8467126.

[6] 6. M. Marques, S. F. Ochoa, M. C. Bastarrica and F. J. Gutierrez, "Enhancing the Student Learning Experience in Software Engineering Project Courses," in IEEE Transactions on Education, vol. 61, no. 1, pp. 63-73, Feb 2018, doi: 10.1109/TE.2017.2742989.

[7] Z. Kubincová and I. Csicsolová, "Code Review in High School Programming," 2018 17th International Conference on Information Technology Based Higher Education and Training (ITHET), 2018, pp. 1-4, doi: 10.1109/ITHET.2018.8424617.

[8] T. Brown, M. R. Narasareddygari, M. Singh and G. Walia, "Using Peer Code Review to Support Pedagogy in an Introductory Computer Programming Course," 2019 IEEE Frontiers in Education Conference (FIE), 2019, pp. 1-7, doi: 10.1109/FIE43999.2019.9028509.

[9] Lazarovych et al., "Software Implemented Enhanced Efficiency BPSK Demodulator Based on Perceptron Model with Randomization," 2021 IEEE 3rd Ukraine Conference on Electrical and Computer Engineering (UKRCON), 2021, pp. 221-225, doi: 10.1109/UKRCON53503.2021.9575458.

[10] M. Kozlenko, O. Zamikhovska, V. Tkachuk and L. Zamikhovskyi, "Deep Learning Based Fault Detection of Natural Gas Pumping Unit," 2021 IEEE 12th International Conference on Electronics and Information Technologies (ELIT), 2021, pp. 71-75, doi: 10.1109/ELIT53502.2021.9501066.