Teaching Distributed Software Architecture by Building an Industrial Level E-Commerce Application

Bingyang Wei, Yihao Li, Lin Deng and Nicholas Visalli

Abstract Software architecture design is a key component in software engineering. Teaching it effectively requires instructors come up with challenging real world projects, in which students need to recognize the architectural problems, figure out solutions, compare alternatives and make decisions. On the other hand, keeping students motivated and interested in this topic is also critical. However, truly understanding and appreciating a certain software architecture are not easy for most undergraduate students. Most architecture courses either focus too much on the theoretical concepts or fail to immerse students enough in complex industrial level projects unless the instructors are practicing software engineers. In this paper, we propose a Project-Based Learning experience, which brings an open-source full-fledged system to classroom in order to effectively teach distributed software architecture. Students are introduced the best practices that are widely used in industry to solve some of today's common architectural problems. Our ultimate goal is to establish a public code repository of realistic projects from popular industrial sectors, so that instructors can reference to improve software architecture's learning experience even though they are not practicing software engineers.

Keywords Distributed architecture · Microservices · DevOps · Software engineering education

B. Wei (✉)
Department of Computer Science,
Texas Christian University, Fort Worth, TX, USA
e-mail: b.wei@tcu.edu

Y. Li
Institute for Software Technology,
Graz University of Technology, Graz, Austria
e-mail: yihao.li@ist.tugraz.at

L. Deng · N. Visalli
Department of Computer and Information Sciences,
Towson University, Towson, MD, USA
e-mail: ldeng@towson.edu

N. Visalli
e-mail: nvisal1@students.towson.edu

R. Lee (ed.), *Software Engineering Research, Management and Applications*,
Studies in Computational Intelligence 845,
https://doi.org/10.1007/978-3-030-24344-9_3

1 Introduction

A software architecture course teaches students system thinking skills. In a typical architecture course, students are taught numerous theoretical concepts and principles; simple example software systems are often introduced with the purpose of motivating students, providing some context, and making students link what they learned to the "real world." [1] Unfortunately, due to the page limitations of the architecture textbooks, most example systems are really toy systems, and too simple to express enough architectural issues [2]. This contradicts with the requirements that training software architects needs to contend with the problem of how to make the learning realistic. For students without any industrial experience, it is a frustrating learning experience since everything is dry and abstract. From the instructors' perspective, many young faculty who just spent the last 5 years working on their Ph.D. may have little experience in industrial software development. The industry relevance and alignment need to be improved.

Our concern is backed by the feedback of graduates from the authors' universities. Graduates reported that many popular architectural solutions that have already become the de facto standard in industry are never taught thoroughly at school, and the teaching materials are seriously outdated. In other words, the gap between undergraduate software engineering education and the industry is becoming larger and larger. As educators, we need to align students' academic knowledge to industrial settings so that they can transition to their first job smoothly.

In this paper, we present a Project-Based Learning (PBL) approach [3] to teach concepts in distributed software architecture. Using this approach, students can learn a step-by-step process of developing a real industrial-level E-commerce system as each architectural concept is taught. The E-commerce project chosen was originally designed by www.itcast.cn for an intensive 3-week software architects training camp. We modified and extended it to fit a one-semester undergraduate architecture course. Our goal is: for every theoretical concept and principle that students learn from the lecture, it is supported by a concrete application scenario in this project. By doing so, our students know why, when and where to apply the learned concept and principle. Furthermore, the course materials will be reviewed by our Strategic Industrial Partners/Industrial Advisory Board every year to make sure that all the techniques and solutions that are used in this project are state of the art and used ubiquitously in industry. We hope this paper can help and inspire instructors during their preparation of software architecture related courses.

The rest of the paper is structured as follows. Section 2 discusses the rationale behind the selection of an E-commerce system as example system. The architectures of the E-commerce system are described in detail in Sect. 3. In Sect. 4, common problems that a future software architect would encounter and the state of the art solutions are presented. Section 5 presents the deployment and testing aspect of this project. Coverage of various of architectural issues, a possible course schedule and future work are discussed in Sect. 6. In Sect. 7, we conclude the paper.

2 The Rationale Behind E-Commerce Systems as a Pedagogical Tool

During the preparation of this course, problem domains familiar to students were given higher priority when it came to selection of example projects. E-commerce is one of the systems that fit this criterion. It is defined as the selling and buying of products (including services) through the Internet. A further look into E-commerce would reveal three main branches: business to consumer (B2C), business to business (B2B) and consumer to consumer (C2C). We chose B2C since it is the most well-known type of E-commerce. Students can easily understand many requirements and problems based on their everyday online shopping experience on Amazon or Best Buy.

The second reason to introduce E-commerce to classroom is the amazing growth of this field. Based on the predication by Statista.com, the annual growth rate of E-commerce in the next 4 years is around 14% [4]. Students are motivated to learn because of the huge job market for E-commerce software developers.

The third reason of using an E-commerce project is that it is a perfect example to reveal issues of various quality attributes of a software system, such as availability, security and scalability. As a matter of fact, modern E-commerce systems are featured by their high concurrency and high availability. During the beginning of this course, we used www.Taobao.com, a Chinese E-commerce giant as an example: Since 2009, www.Taobao.com holds the Single's Day Online Shopping Event on every November 11th. With four characters of ones, this date is called Single's Day. In fact, everyone, not just single people will take advantage of this one-day sale and purchase online. Here are some amazing statistics about this 24-h sale in 2017 that we share with students (see Fig. 1).

Based on the statistics released by www.Taobao.com [5], the Gross Merchandise Volume (GMV) reached at 1.47 billion dollars in the first three minutes of November 11th, 2017. By the end of the day, the GMV was over 26 billion dollars. The total number of payment transactions is 1.48 billion, and the number of orders placed and processed is 812 million. In the first 5 min of the sale, the payment software (AliPay) used by Taobao was handling 256,000 transactions per second. The most amazing part of this examples is, the system functioned perfectly on that day, without any downtime. Based on the feedback after class, students were shocked by this super project that could be so available and reliable under so much traffic. Students are highly motivated to keep on learning the concepts and principles behind this success.

Fig. 1 Timeline of the 2017 Single's Day Sales in million US dollars

In summary, we believe that an industrial level E-commerce system can both motivate and interest students in terms of its promising job market and various technical aspects.

3 Architectures of E-Commerce Systems

This course is designed for senior computer science students as an elective course. They should already have some experience in developing small scale web or mobile applications. The course starts with introducing the traditional architecture for building a web application, and then discusses the drawbacks and possible improvements. The distributed architecture is introduced later.

3.1 Traditional Architecture

Traditional architecture of a web application includes every module into a monolithic project (see Fig. 2). For example, in our E-commerce system, one project is developed for online shoppers to browse, search, add to cart, log on and check out. Those functionalities are included in the project called, "Online Shopping System," as modules (i.e. Java packages). Another project is designed for the administrators to maintain the website. For example, addition, removal, modification of a product, creation of advertisement and promotions. These two monolithic projects are then deployed to a single web application server. For small traffic, one server might be enough. But when the application server is receiving a lot of traffic during holiday season, the projects will need to be duplicated to more servers (see Fig. 3). This clustering approach is easy to understand, develop and deploy. Students in our class already have some experience in using this kind of architecture.

The traditional architecture's drawbacks are then explained to students. First, not every module in the "Online Shopping System" receives the same amount of concurrency or pressure. For example, there will always be more people browsing and searching than really buying a product. The Administration System's pressure is generally orders of magnitude smaller than the "Online Shopping System," since its users are just a dozen of administrators and IT operation personnel. So, it is a waste to deploy everything on multiple application servers at the same time. This leads to poor scalability, and what we really should do is to add more servers to extend the services that are under most pressure. The second drawback is that such an architecture makes it hard for a team to develop. Different teams' work needs to be integrated into one project in order to build and run. For example, if the frontend UI team only needs to modify the page of a product (maybe a typo), it needs to repackage and redeploy the entire application (during which period, the entire website is down).

Once the pros and cons of the traditional architecture are explained. We can move on to the distributed architecture.

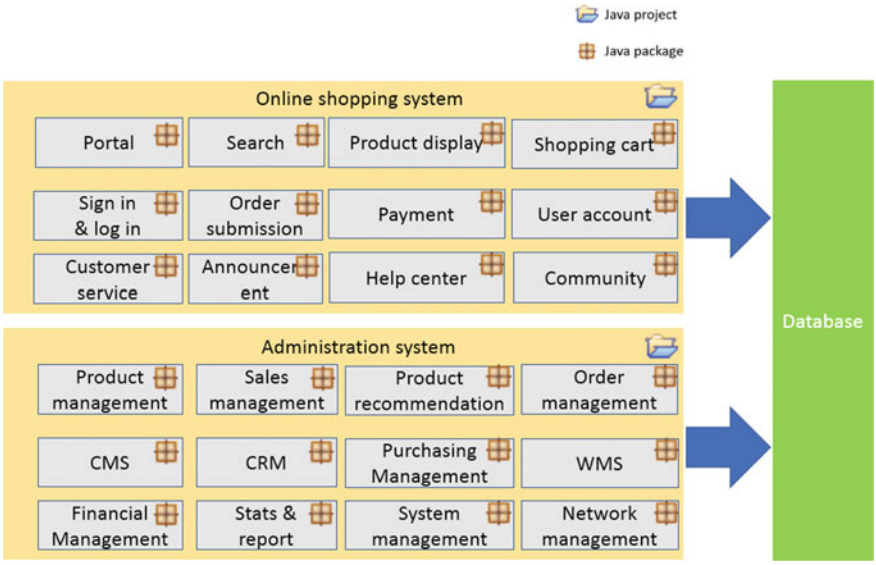

Fig. 2 Monolithic E-commerce software system architecture

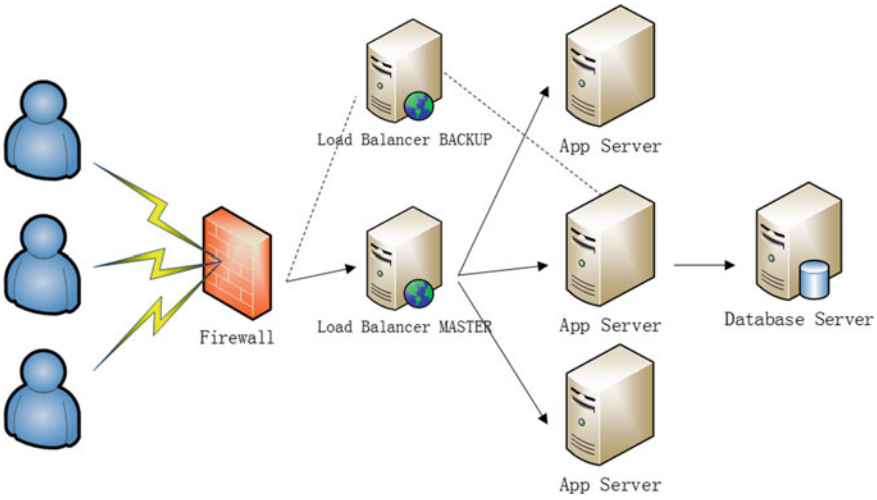

Fig. 3 Traditional network topology for E-commerce systems

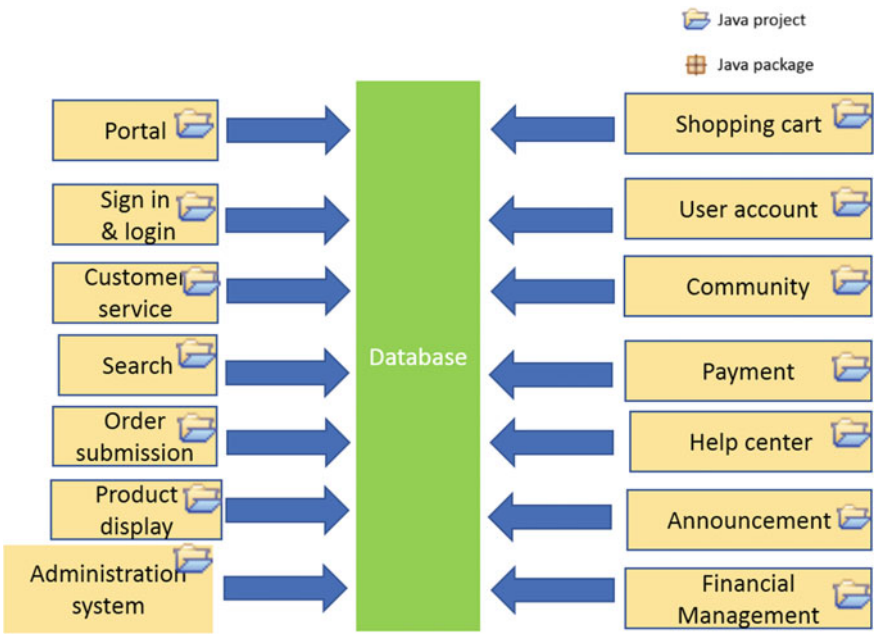

Fig. 4 Major components of the system

3.2 Distributed Architecture

Based on the analysis of the traditional architecture, students are encouraged to figure out ways to correct the architectural issues. Every module from the traditional architecture should be taken out from the monolithic system, and an independent system is developed for it (see Fig. 4). Each module will run in a separate Docker container and communicate to other modules through RESTful web services. Separating out these modules can also facilitate better team collaboration and project management.

4 Key Solutions for Achieving High Availability and Concurrency

As a software architect, one important trait is to stay current with mature solutions and know how to make tradeoffs. Based on our research of the current technology trends and the conversions with industrial partners, a dozen of mature solutions for achieving system availability and concurrency are identified and taught to students (Table 1). We want to warn the readers that this is an "opinionated" technology stack. They represent typical and contemporary industrial distributed software development challenges. The frameworks used for our E-commerce system is SpringMVC + SpringBoot + SpringData + SpringCloud.

Table 1 Solutions for common problems in distributed system development

Problems	Solutions adopted
Dev, test, staging and production environment switches	Spring profile
Coordination of distributed systems	Spring cloud
Distributed ID generation	Twitter IDs (snowflake)
Authentication and authorization	Spring security and JSON Web Token (JWT)
Caching	Redis
Job scheduling	Quartz
Search engine	Solr
Load balancing	Nginx
Data access	Spring data
Message broker	RabbitMQ
Distributed file system	FastDFS
Deployment	Docker and Rancher

The solutions mentioned in Table 1 should not be introduced at the very beginning of the project development. For example, the best practices to solve problems like distributed ID Generation, Authentication, Single Sign On, Microservices Management, Database Clustering, Template Engine, Distributed File System are not revealed in the first increment of the E-commerce system. Instead, students are encouraged to solve those problems by pure Java code. This is a valuable experience, since the first increments becomes the reference point for the future iterations. In the following iterations, mature solutions are introduced to students one by one, and refactoring takes place. Besides the solutions "adopted" by this course, comparing alternatives is assigned as homework. For example, for the problem of Search Engine, students also try ElasticSearch; for Message Broker, Kafka is also tried and compared.

The architecture of the E-commerce system keeps evolving throughout the entire course. Figure 5 describes the final version of the architecture at the end of this course. Due to the space limitation, not all the techniques are shown in the figure.

5 Continuous Integration, Deployment and Testing

Deployment and operations are also designed to be part of this course. During the first half of the course, virtual environment VMWare is used to simulate the deployment. We then switch to Amazon AWS deployment to make the deployment real. We also introduce the DevOps approach. Based on Fig. 6, the students are taught to provision Amazon AWS servers to host Jenkins and Artifactory, respectively. Jenkins is one of the automation tools for facilitating continuous integration and continuous delivery, while Artifactory is a popular enterprise artifacts management tool. Jenkins and

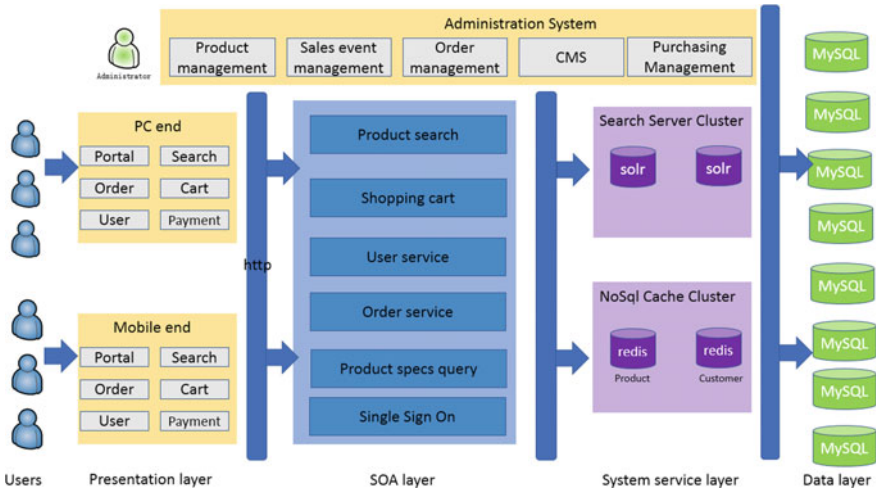

Fig. 5 Distributed E-commerce software system

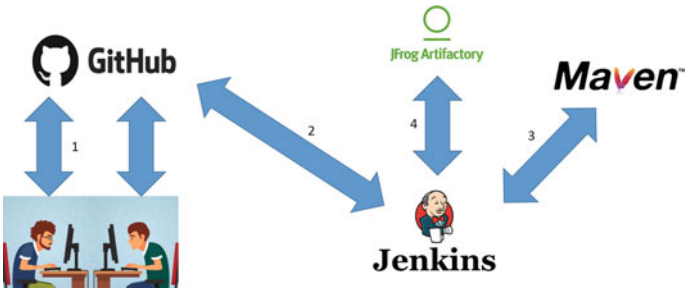

Fig. 6 Continuous integration technology stack

GitHub are connected through web hook so that every time there is a commit to GitHub, Jenkins can pull the artifacts from both GitHub and Artifactory, build and test them. During this course, every service is running in a Docker container.

The development of the E-Commerce system follows the Test-driven development (TDD) approach. Since the project is designed around the MVC layered architecture. Unit tests for service layer and controller layer are written first; code for service and controller layers are added in order to pass the unit test. Integration test that involve database and more external resources are written to make sure the modules work well together. For now, JUnit, Mockito and Spring MVC Test are used in the project.

6 Discussion

In this section, the mapping between desired virtues of a qualified software architect and our proposed course, quality attributes covered in the course, potential schedules and future work are discussed.

6.1 Coverage of the Desired Virtues for a Software Architect in This Course

Bass and Clements [6] list the traits of a qualified software architect in their architecture book. Table 2 describes how our teaching approach checks off each of them.

Table 2 Developing the desired virtues for a software architect

Traits of software architects	How our proposed course is related to training the traits
Artistic skill to make seemingly simple designs that are easy to grasp and yet which solve complex problems	We bring many common problems (see Table 1) and ask students to use different techniques to solve them
Analytical skills, such as an ability to find root causes for high-level problems in existing designs, like why a system runs too slowly or is not secure	Deep analysis of the drawbacks of the traditional architecture for E-commerce system and comparison of different alternatives for solving the same problem
Understanding of the business, social, and operational environment in which a system needs to operate	Students are taught E-commerce domain knowledge along the way. Students are trained to operate under development, testing and production environments
Ability to relate to a wide range of stakeholders, including the client, users, system administrators, and suppliers	During design, students are taught to think from different perspectives to make design decisions
Communication skills - such as listening to and working out issues with those implementing a design	Students work in a team to solve problems together
Knowledge of multiple technologies, from which to choose the right ones for the next job	A dozen of current technologies are taught to students, comparisons are required in assignments

6.2 Coverage of Quality Attributes in Our Proposed Course

The Key Considerations in Teaching Software Architecture CSEE&T 2004 workshop pointed out the emphasis of a software architecture course needs to be on the quality attributes (QAs). We list a set of QAs and how our course is designed to cover them in the E-commerce project (Table 3).

6.3 Related Work

Using huge and open-source systems as a tool for students to work on and make architectural decisions is discussed in the work of Costa-Soria and Perez [7]. In their course, students are asked to reverse-engineer open source projects and write reports about the architecture used. Our approach, however, is to train students to build a large, complex application from scratch using the latest technology. This process is guided by the instructor who is very familiar with the domain and current technology. In fact, the development of this realistic E-commerce systems requires extensive work by both students and instructors. We describe a possible schedule in the next section.

6.4 Suggested Course Schedule

The content in this paper can be taught as a standalone undergraduate senior course. Table 4 shows a possible schedule for a one-semester hands-on course. This schedule is based on the assumption that students already have experience in web application development. Every week, there are dedicated lab hours for students to finish the milestone so that students can build the project incrementally and no one is left behind. Additional assignments for comparing solution alternatives are given every week.

Table 3 Quality attributes considered by students in this course

Quality attributes	How our proposed course covers
Availability	Clustering, master/backup servers
Performance	Additional layer of cache between app and database, e.g. Redis and Solr, and Database Sharding
Security	Spring security and JWT
Scalability	Microservice approach, e.g. spring cloud
Interoperability	RESTful web service, JSON-based communication
Maintainability	Independent microservices and use of DevOps

Table 4 Schedule of the E-commerce course

Weeks	Contents
Week 1	Background of E-Commerce. Introduction of distributed architecture
Week 2	Integration of Spring, SpringMVC and hibernate for the administration system and online shopping system
Week 3	Implementation of the administration system
Week 4	Implementation of the online shopping system
Week 5	Implementation of CMS (Content Management System)
Week 6	Adding cache using Redis
Week 7	Implementation of searching using Solr
Week 8	Implementation of single sign on subsystem with session sharing and spring security
Week 9	Implementation of shopping cart and order subsystems
Week 10	Amazon AWS, Docker and Artifactory
Week 11	Deployment in distributed environment (Nginx and reverse proxy)
Week 12	Set up Redis cluster and Solr cluster
Week 13	Refactor using spring family (SpringBoot, SpringCloud, Spring, SpringMVC and Spring Data)

6.5 Future Work

An important future work is to modify more industrial level projects that we already collected from our partners in domains like Online Education, Online Social Network, Online Health, and Online Tourism. We firmly believe that bringing realistic and interesting domain knowledge into classroom generate a good context for students. A repository of large and industrial level open source projects with detailed instructions will be created on GitHub.

7 Conclusion

In this paper, we report the design of a project-based learning approach, which makes use of an example from industry in an undergraduate software architecture course. By using a concrete and realistic case study from a familiar area, students are given better context to apply the architectural principles learned from lecture. Students can learn about new technologies by systematically evolving their own systems. The primary focus of this course is on quality attributes such as high availability and high scalability. Many current mature solutions are introduced so that students will be in a better position during job interview. We hope this paper can help instructors who want to bring realistic projects to architecture class.

References

1. Van Deursen, A., Aniche, M., Aué, J., Slag, R., De Jong, M., Nederlof, A., & Bouwers, E. (2017, March). A collaborative approach to teaching software architecture. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 591–596). New York: ACM.
2. Rupakheti, C. R., & Chenoweth, S. (2015, May). Teaching software architecture to undergraduate students: An experience report. In *Proceedings of the 37th International Conference on Software Engineering* (Vol. 2, pp. 445–454). New York: IEEE Press.
3. Bender, W. N. (2012). *Project-based learning: Differentiating instruction for the 21st century*. Corwin Press.
4. Retail e-commerce sales in the United States from 2016 to 2022. https://www.statista.com/statistics/272391/us-retail-e-commerce-sales-forecast/.
5. Alibaba smashes its Single's Day record once again as sales cross $25 billion. https://techcrunch.com/2017/11/11/alibaba-smashes-its-singles-day-record/.
6. Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley (Professional).
7. Costa-Soria, C., & Pérez, J. (2009, July). Teaching software architectures and aspect-oriented software development using open-source projects. In *ACM SIGCSE Bulletin* (Vol. 41, No. 3, pp. 385–385). New York: ACM.