Software System Design

16-lectures course presented by Yegor Bugayenko
to 3rd-year BSc students of Innopolis University in 2021,
and video recorded

The entire set of slide decks is in yegor256/ssd16 GitHub repository.

Introduction

Who is the teacher? I'm not a professional teacher and, frankly speaking, I'm doing this course as a humble attempt to become one. My background is 25+ years of practical software engineering in small startups and large software companies, like Huawei, where I'm at the moment a director of an R&D laboratory focused on software quality problems. You may find some lectures I've presented at some software conferences on my YouTube channel.

Why this course? The idea to present this course organically came out from cooperation, which has recently started between Huawei and Innopolis University. We both realized that the practical experience Huawei has gained by writing millions line of code over the last few decades may be a valuable contribution to the education provided by the University to the next tech generation.

What's the methodology? The learning process in this course is very much *problem-based*: each lecture presents a number of key problems software development industry has known for years. For example, in the lecture about database design I demonstrate that transaction isolation, schema versioning, data distribution, and performance are the problems. Then, I explain some solutions which already exist and suggest future research directions to find better ones. On top of that, the learning is *project-based*, since it's required for the students to apply some of the solutions suggested in the projects they are making in order to pass the exam at the end of the course.

Course Aims

Prerequisites to the course (it is expected that a student knows this):

- How to code
- How to use Git

After the course a student *hopefully* will know:

- How to manage software requirements
- How to develop iteratively and incrementally
- How to think with objects, not procedures
- How to use design patterns and not use anti-patterns
- How to draw and share knowledge using UML
- How to choose and use data formats, e.g. XML or JSON
- How to choose a database management server
- How to deploy software continuously
- How to build distributed software systems
- How to test software
- How to measure the quality of software design

Assessment

At the end of the course a student receives a *score* of up to 100 points. The points are given after a *subjective* review of an open source software product created by the student during the course (no oral presentation is needed). Even though the review is subjective, the following balance has to be maintained (the questions provided below stand merely as examples and do not constitute the entire scope):

- **Requirements** (15%): Glossary is in place? Stakeholders and their concerns are identified? Use cases explain functional requirements? Non-functional requirements are documented? NFRs are measurable?
- **Design** (25%): UML diagrams, such as Class, Component, Deployment, and Sequence, are present? Design decisions are explained? Design patterns are used? Traceability between requirements and design elements is visible?
- **Architecture** (30%): The design is modular? The composition of modules makes sense? Design elements are cohesive? Design elements are decoupled enough? The build is automated? The delivery pipeline is automated?
- **Code** (15%): The code is clean enough? In-code documentation is present? Static analyzers and style checkers are used? Unit tests are in place? Integration tests are present? Is test coverage being measured?
- **Spirit** (15%): The product is somewhat popular on GitHub (or a similar platform)? Issues and pull requests were used during development? Commit comments are detailed enough? GitHub features are actively used, like releases, actions, etc.?

A few versions of the product may be presented for review: Alpha, Beta, and Final. The scores given to a student after version reviews don't affect the overall score given at the end of the course. However, if Alpha version is not delivered, a student gets a penalty of 10 negative points, while a missed Beta gives 20 negative points. Thus, if a student ignores both versions and brings a great product at the end of the course, he or she gets $100 - 30 = 70$ points at most.

The score may be turned into a grade using the following formula:

- **A** Excellent: 90+
- **B** Good: 75+
- **C** Satisfactory: 55+
- **D** Poor: 0+

Learning Material

The following books are highly recommended to read (in no particular order):

Len Bass et al., *Software Architecture in Practice*

Paul Clements et al., *Documenting Software Architectures: Views and Beyond*

Karl Wiegers et al., *Software Requirements*

Alistair Cockburn, *Writing Effective Use Cases*

Steve McConnell, *Software Estimation: Demystifying the Black Art*

Robert Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*

Steve McConnell, *Code Complete*

Frederick Brooks Jr., *Mythical Man-Month, The: Essays on Software Engineering*

David Thomas et al., *The Pragmatic Programmer: Your Journey To Mastery*

Robert C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*

Grady Booch et al., *Object-Oriented Analysis and Design with Applications*

Bjarne Stroustrup, *Programming: Principles and Practice Using C++*

Brett McLaughlin et al., *Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D*

David West, *Object Thinking*

Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*

Yegor Bugayenko, *Elegant Objects*

Michael Feathers, *Working Effectively with Legacy Code*

Martin Fowler, *Refactoring: Improving the Design of Existing Code*

Erich Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*

Scott Meyers, *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*

Elliotte Rusty Harold et al., *XML in a Nutshell, Third Edition*

Michael James Fitzgerald, *Learning XSLT: A Hands-On Introduction to XSLT and XPath*

Martin Fowler, *UML Distilled*

Anneke Kleppe et al., *MDA Explained: The Model Driven Architecture: Practice and Promise*

C.J. Date, *An Introduction to Database Systems, 8th Edition*

Pramod Sadalage et al., *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*

Jez Humble et al., *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*

Michael T. Nygard, *Release It!: Design and Deploy Production-Ready Software*

Leonard Richardson et al., *RESTful Web APIs: Services for a Changing World*