Ensuring Quality in Software Projects

Series of lectures by Yegor Bugayenko

Abstract:
The course is a series of loosely coupled pieces of advice related to quality of software development. Pragmatic programmers may listen to them if they don't want to tolerate chaos in their projects. The course is not only about coding practices, but also about static analysis, test coverage, bug tracking, dependency and artifact management, build automation, DevOps, and many other things. If we don't do them right, they may severely jeopardize the quality of the entire project, no matter how good are your algorithms.

Who is the teacher? I'm developing software for more than 30 years, being a hands-on programmer (see my GitHub account: @yegor256) and a manager of other programmers. At the moment I'm a director of an R&D laboratory in Huawei. Our primary research focus is software quality problems. You may find some lectures I've presented at some software conferences on my YouTube channel. I also published a few books and wrote a blog about software engineering and object-oriented programming.

Why this course? In one of my videos a few years ago I explained what I believe is killing most software projects: it's the chaos they can't control. Most of us programmers start projects full of enthusiasm and best intentions. We are confident that this time the design will be solid, the code will be clean, and our customers will be happy because there will be no bugs. Eventually, sooner rather than later, the reality appears to be as bad as it was in the previous project: the code is messy, the design resembles spaghetti, and the bugs are unpredictable and hard to fix. We learn some lessons, abandon the project, and start a new one, again with the best intentions. But in a new project nothing changes. Most programmers that I know run in this cycle for decades. I believe, this course may help you not become one of them.

What's the methodology? The course is a collection of individual cases not closely connected to each other. Each lecture discusses a single open-source GitHub repository. Each discussion highlights the mistakes made in the repository and suggests improvements. Each lecture ends with a conclusion and a formulated recomendation. The recomendations may help students prevent and control chaos in their own future projects.

Course Structure

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

- How to use Git
- How to code
- How to design software
- How to write automated tests
- How to deploy

After the course a student *hopefully* will understand:

- How to avoid code smells
- How to convince a manager that quality is important
- How to deal with negligence of other programmers
- How to hire a programmer who cares
- How to argue with customers about quality
- How to discipline fellow programmers
- How to protect yourself from chaos

Also, a student will be able to practice:

- Source control: Git, Subversion
- Build automation: Make, Maven, Gradle, Grunt, Rake
- Dependencies: Maven Central, NpmJS, RubyGems, PyPi
- Static analysis: Clang-Tidy, SpotBugs, Coverity
- Style checking: Checkstyle, PMD, Rubocop, Eslint, Qulice
- Automated tests: JUnit, Mocha
- Integration tests: Cucumber, Selenium, Cross-Browser
- Mocking frameworks: Mockito, PowerMock
- Textual documentation: Markdown, Wiki, LaTeX, CNL
- Bug tracking: GitHub, JIRA, Bugzilla
- Code reviews: GitHub, Gerrit, Crucible
- Test coverage: JaCoCo, Codecov
- Mutation coverage: PIT
- Property-based testing: Quickcheck
- DevOps: Docker, Heroku, AWS
- Pre-flight builds: GitHub Actions, Jenkins, Rultor
- Metrics: SonarQube, CodeClimate, jPeek.

Lectures

This is a list of cases that will be discussed at the lectures:

1. PMD rules in yegor256/qulice
2. Managing Maven dependencies in yegor256/takes
3. Multi-module `pom.xml` in objectionary/eo
4. Deploying to Maven Central in yegor256/cactoos
5. Deploying Java app to Heroku in zerocracy/farm
6. Code reviews in objectionary/eo
7. Making a Ruby gem in zold-io/zold
8. Deploying a Ruby web app in yegor256/sixnines
9. Npm dependencies and Grunt in objectionary/eoc
10. Build automation with `Makefile` in yegor256/fibonacci
11. Build automation with Gradle in objectionary/eo-intellij-plugin
12. Ruby style checking with Rubocop in yegor256/sibit
13. Integration testing against DynamoDB Local in yegor256/rultor
14. JaCoCo coverage control in yegor256/cactoos
15. Fake GitHub and AWS S3 objects in jcabi/jcabi-github and jcabi/jcabi-s3
16. Packaging for CTAN in yegor256/ffcode
17. Docker image releasing to the Hub in yegor256/rultor-image
18. Parametrized testing with YAML in objectionary/eo
19. GitHub actions in yegor256/fibonacci and jcabi/jcabi-xml
20. Integration testing with Maven Invoker Plugin in jcabi/jcabi-xml
21. CNL for requirements specification in yegor256/requs
22. Making and testing a new GitHub Action in yegor256/latexmk-action
23. XML style checking in yegor256/xcop
24. Headless in-browser testing with Selenium in yegor256/jare
25. Reversive deployment to AWS EC2 in yegor256/s3auth
26. Hits-of-code and other metrics in yegor256/cobench and yegor256/hoc
27. Testcontainers in yegor256/threecopies
28. BDD with Cucumber in cqfn/pdd
29. Calculating Java cohesion metrics in cqfn/jpeek
30. Documenting `README.md` of a Java library in yegor256/takes

Students are welcome to pick most interesting cases.

Laboratory Classes

A few following laboratory classes may support the course, where students will be asked to solve some of these tasks (the most complex are at the bottom):

1. Configure GitHub Action to publish code coverage to codecov
2. Make both JUnit4 and JUnit5 tests work inside one Java repository
3. Create a new unit test in BDD style, using Cucumber or a similar framework
4. Design a coding style guide (in Markdown) for your favorite language
5. Configure GitHub Actions to deploy a new JavaScript library to npmjs
6. Automate headless Selenium+Safari integration testing in a simple web app
7. Automate mutation coverage publishing to GitHub Pages
8. Configure build to fail if test coverage is lower than 80%
9. Create a new plugin for GitHub Actions to validate the layout of repository
10. Merge a pull request improving coding style to a 10k+ GitHub library
11. Create an automated test for Java+Hibernate+MySQL with testcontainers
12. Create GitHub Action to spell check `README.md` using aspell
13. Install Jenkins and configure it to merge branches on demand
14. Configure GitHub Action to send a message to Telegram when the build fails
15. Automate cross-browsing testing of a web app using BrowserStack
16. Configure property-based testing in an existing repository
17. In a existing Java project automated with Maven, replace Maven with Makefile
18. Automate comparison of two static analyzers, find out which one is stronger
19. Develop a Maven plugin to check the quality of POM file
20. Create a new style checking rule for PMD, to prohibit the use of non-final classes
21. Using aspell, create a style checker validating grammar in Java comments

There could be other tasks too.

Grading

Students may form groups of up to four people. Each group will present their own public GitHub repository with a software module inside. The group will make a presentation of the quality control mechanisms that are present in the repository. They will have to explain during a 10-minutes oral presentation with live GitHub demonstration via screen sharing:

- How enabled quality ensuring mechanisms work?
- Why such mechanisms are in use?
- How they help ensure quality?
- How often they get activated?
- What are the drawbacks of them?
- What mechanism are not used and why?

Most probably, there will be no more than 20% of "A" marks, no more than 40% of "B," and the rest will go to "C" and "D." However, this distribution is not mandatory: if all students make excellent presentations, everybody will get "A."

The teacher of laboratory classes may increase or decrease the grade by one point.

Higher grades will be given for:

- Better understanding of the reasons behind used mechanisms,
- How they help ensure quality,
- How often they get activated, and
- What are the drawbacks of them.

A retake exam is possible, following exactly the same procedure. However, the highest mark most probably possible at the retake is "C."

Students are highly advised to discuss their repositories and quality ensuring mechanisms with each other, before the final exam, in order to understand their relative positions and maybe trigger new ideas.

Learning Material

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

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*

David West, *Object Thinking*

Yegor Bugayenko, *Code Ahead*

Michael Feathers, *Working Effectively with Legacy Code*

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*