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.

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 Aims

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 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
- Textual documentation: Markdown, Wiki, LaTeX, CNL
- Bug tracking: GitHub, JIRA, Bugzilla
- Code reviews: GitHub, Gerrit, Crucible
- Test coverage: Jacoco, Cobertura
- Mutation coverage: PIT
- DevOps: Docker, Heroku, AWS
- Pre-flight builds: GitHub Actions, Jenkins, Rultor

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?

Groups will compete with each other for the grades. Totally, there will be no more than 20% of "A" marks, no more than 40% of "B," and the rest will go do "C" and "D." 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.

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 higher mark possible at the retake is "C."

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*