MUNI
FI

PV260 – Seminar 2
Code Review, Continuous Integration

18. 2. 2022

Matěj Karolyi

Terms and Conditions

- This PDF file is available only for students of the PV260 Software Quality course for learning purposes only.

- Please, do not modify or redistribute this PDF under any circumstances.

Today's topics

- Part 1: Reviews
  - Motivation
  - Practical showcase
  - Two exercises
- Part 2: Continuous Integration
  - Introduction of the concept
  - Jenkins
  - Practical showcase
  - Two exercises

MUNI
FI

Reviews

Part 1

Reviews

– Taking look at the work of others. There are different types of reviews.

1. Code Review
2. Sprint Review
3. Design/Specification/Requirements Review
4. Test Plan Review
5. Employee Review by Management
6. "Email written with anger" Review before clicking Send button :)
7. ...

Exercise #1

- Why are the reviews important? In 5 minutes, try to figure out how reviews can be useful (either some specific review or reviews in general). Is it even possible for a team to work without reviews?
- NOTE: You will remember the answers better if you think about them and realize them yourself than if told/read somewhere.
- Write your answer to the Socrative.

Exercise #1 – solution

1. Find problems/possible improvements early (incomplete design/specification; bug in the code; not very useful employee; missing important areas in the test plan; ...)
2. Knowledge sharing / Learning / Inspiration (both sides on the Review can learn from each other; or together they might find out what none of them knows so far)
3. Team-building (people understand each other better and can better communicate among each other)
4. Motivating others (for shy/not very self-confident employees might be important to hear they did a good job and it may improve their work)
5. Improve processes (how to be more effective at sprint reviews/retrospectives)
6. Make you come out of daily routine where you are mostly concerned on small details and make you think about the big picture for a while

Types of code review

1. Formal vs Informal
2. Documented vs Non-Documented
3. Lead by Author of the change vs Lead by trained moderator
4. By the main purpose: Find bugs only? Learn something? Evaluate alternatives? Solve technical problems? Gain understanding? ...
5. Pair programming
6. Tool-assisted or not

Lessons learned

- What are reviews and that they can be used for basically anything.
- Various benefits of using 'review' approach.
- That reviews can be pretty cheap (time and money) and also quite expensive.

Reviews on GitHub

- Built-in support of reviews
- Automated checks
- Symfony open-source project community contributions
  - https://github.com/symfony/symfony/pulls

Exercise #2

- In 3 minutes think on your own how the proper code review should look like. What attributes it should have and on the other side what the reviewer should avoid.
- In next 7 discuss it with your “assignment colleague” in breakout room.
- Write down your answers to the Socrative.

MUNI
FI

Continuous Integration

Part 2

CI – introduction

- A lot of people working on one product. They want to commit often. This can very simply lead to problems called integration hell. So the problems should be detected and fixed as soon as possible. That requires a lot of automated test and also automated running of the automated tests.

CI – definition

- **#1**: Continuous Integration is the practice, in software engineering, of merging all developer working copies to a shared mainline several times a day. [Source: Wikipedia]
- **#2**: Continuous Delivery is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. [Source: Wikipedia]

CI systems – definition

- **#1**: Continuous Delivery System is a software which support CI.
- **#2**: Build is a set of tests defined by some rule which should be run together on a branch.
- **#3**: Build run is a deployment of a software on a specified version of a branch and running the tests specified by the build.

Exercise #3

- In 10 minutes, try to think of features a CI system should have and list them.

- Write suggested features to the Socrative.

Exercise #3 – solution (integration)

1. Continuous Delivery
2. Build system
3. Version Control System
4. Issue Tracking System
5. IDE
6. Email service
7. Code Review System

Exercise #3 – solution (configuration)

1. Hardware setup (machines where software will be deployed and tests run)
2. Allowing pre-commit (change is not integrated into branch until all relevant tests are executed and passed)
3. Environment setup (java version, variables, maximum time of build run, ...)
4. Easily configurable builds (which tests they include) on every branch. Configuration of events which trigger a build run of a build.
5. Possibility to lock branches (integration with VCS)

Exercise #3 – solution (view)

1. Test History + visible what CL have been submitted in each build
2. Screen-shots from failed system tests if possible
3. View of all builds (with all relevant tests) for a submitted CLs. New failing tests should be displayed.
4. Visibility of specific run of a build - failing tests, new failures, tests which already passed in later build runs, passed tests, environment setup used, artifacts, CLs submitted from previous build run, execution log, ...
5. Configurable homepage with links to specified builds

Exercise #3 – solution (others)

1. Test/Build Investigation possibility (so that others know what's happening)
2. Possibility to mark the test "Muted" and "Intermittent"
3. Email send to those who broke the build or notification that build passed after your commit
4. Search (either a test, CL, branch, person, ...)

Jenkins

- “An open source automation server which enables developers around the world to reliably build, test, and deploy their software.”
- “The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.”
- CI, CD, Plugins, Extensions, Test automation.

Exercise #4

- In 5 minutes, try to think of possible problems and challenges of CI in big companies.

- Write down your answers to the Socrative.

Exercise #4 – solution

1. Clean Up (preparing DB from DB dump after each test/test class might be too time consuming). Some tests might be written that they do not do proper clean up.
2. Long execution time of build runs. When do you want to know whether the change is OK or not? Minutes? Hours? Days? How to make it so that you know result soon enough?
3. Running only tests which can become broken after a commit.
4. Broken branches for too long. Solutions: pre-commits; automatic reverts; locking branch and only some specific commits allowed.
5. Intermittent tests - failing build runs when they are good – a lot of human time spent on investigation and test fixing.
6. Problems with CI and agents causing intermittent tests.

Exercise #4 – solution

7. Test passes locally but fails in CI.
8. Way too many CLs in one build, hard to identify which one broke the tests.
9. How to write tests so that there are not much false negatives and false positives?
10. What about test cases which are not and/or cannot be covered by automated regression testing and must be verified manually?
11. How to write tests so that their execution time is small?
12. Price of Continuous Delivery (might be quite expensive to run many tests over and over again)
13. How to make test investigation as easy as possible when failure in CI?

Lesson learned

1. You should understand what is Continuous Integration and Continuous Delivery and why/how to use them. You should also have some idea what a CI system should be able to do.
2. You should understand that using CI can also have some negative aspects and that there are lots of non-trivial problems and challenges you need to solve in order to use CI efficiently.
3. CI is basically a must for big (and even medium) software companies nowadays and probably suitable for most of the small companies as well.