

# Continuous Code Quality Testing

Testival #46, Marko Kruljac

“Today, I will create horrible,  
unmaintainable, buggy legacy code”

“Today, I will create horrible,  
unmaintainable, buggy legacy code”

No one, ever

# Why is bad code a problem (for business)?

- Filled with bugs
  - Expensive to correct
  - Not competitive
- Delays launch plans
  - This again costs the company money
- Difficult to change
  - Products that do not adapt get overrun
  - Simple changes become too expensive to implement
- Hard to attract talented engineers
- Even harder to keep good talent

“Indeed, the ratio of time spent reading vs. writing is well over 10:1. We are constantly reading old code as part of the effort to write new code.”

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



To solve the code quality problem, let's  
not talk about code

Which building is more likely to  
deteriorate in condition?







Which cleaning task is easier to complete?







# The problem

- Source code is no different
- Abandoned, bad code gets worse, like abandoned buildings
- Fixing lot of bad code is very difficult, like doing dishes

# The solution?

- Fix house issues as soon as they appear
  - This assumes you are able to detect the issues (automatically or manually)
- Do not abandon the house
  - This sounds simple, but why is it difficult to implement in software?
- Do not let dishes pile up
  - Clean in regular, more frequent cycles
- Static code analysis and Continuous testing

Simply saying “Read books and learn best practice” is not good enough.

Without testing code quality  
continuously, on every change - All code  
converges to legacy



# Static Code Analysis Continuous Testing

Supported with a good code review process

# Static Code Analysis

- The process of testing and analysing source code statically, without running the program
- Static analysis tools analyse the source code against a predetermined and predefined set of rules

# Static Code Analysis CAN detect

- Bad code
  - Copy/pasted, not compliant to style guide, spaghetti, etc.
- Overly-complex code
  - Cyclomatic complexity, too big methods, etc.
- Invalid code
  - Variable types not matching or unreachable code, overflow risks, etc.
- Insecure code
- Performance concerns
- Many other things, limited only by the specific tool you use

# Static Code Analysis key points

- Fast
- Requires no extra time after initial setup
- It helps focus the process of code review, since the reviewer needs only to review parts which were not checked by static analysis

# Static Code Analysis limitations

- Does not find bugs in real, runtime environment
- The tools produce false positives and false negatives
  - “Humans are still smarter than machines”
- Tools are scattered and limited depending on your tech stack

# Continuous Testing

- The process of executing automated tests as part of the software delivery pipeline
- The goal is to obtain immediate feedback on the business risks associated with a software release candidate
- But most importantly, it is a cultural shift

# What do you mean by culture shift?

- Without the right workflow and mindset to back it up, the idea of continuous testing does not work

# Common development workflows

- Level 0: Pushing directly into production



# Common development workflows

- Level 0: Pushing directly into production
- Level 1: Branches and pull requests, code reviews before merging. No tests

# Common development workflows

- Level 0: Pushing directly into production
- Level 1: Branches and pull requests, code reviews before merging. No tests
- Level 2: Tests + Continuous Integration + Pull Request reviews



# Our workflow? No code reviewing if...

- Method has cyclomatic complexity number larger than X
- Method is accepting more than Y parameters
- Method is has more than Z lines of code
- Copy/pasted code
- Commented code
- Code is not formatted according to style guide
- Code does not compile (or pass linter if scripted language)
- **At least one regression test failing**
- **Code coverage from tests is below predefined threshold**

# The result?

- Bad code gets quickly rejected by our CI bot
  - Developers do not waste time reviewing code (which should not pass the review anyway)
- Code review takes much less time, since lot of ground is already covered automatically
  - Reviews can focus on the really important, strategic, architectural matters
- Code is never abandoned and never degrades in quality
- Bad code cannot pile up, since it must be fixed as soon as it is detected

# The catch?

- “Without the right workflow and mindset to back it up, the idea of continuous testing does not work”
- In the short run, it will decrease velocity by some fixed amount
- In the long run, you make up for this loss by being able to refactor code with less friction
  - If your projects do not have a “long run”, you are not getting this benefit - only the cost

Thank you for attending Testival #46  
Time for Q&A