M.EIC, 2022-23

Large Scale Software Development

Filipe Correia, Daniel Pinho, João Pedro Dias

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

DEI
DEPARTAMENTO DE
ENGENHARIA INFORMÁTICA

Pull Requests, Code Review, Continuous Integration and Branching Models

U.PORTO
DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Pull Requests

- Born in the context of open source software development.
- There's three sides to PRs:
  - the **documentation** of the changes.
  - the **asynchronous code review and discussion** of the changes.
  - the assumption of using **feature branches**.

Should Pull Requests be used?

It depends!

(Note: in any case, we will use them for our projects in DS)

Should Pull Requests be used for *documenting* changes?

Documenting changes is usually a good idea...

... even if we don’t necessarily need feature branches to do that.

Should Pull Requests be used for *asynchronous code reviews*?

- **Asynchronous code reviews** are often used by teams in low-trust environments (e.g., open source contributions).
- Co-located teams very often will benefit more from **synchronous code reviews**.
- What does a synchronous code review look like?
  - Pair Programming
  - Ensemble Programming

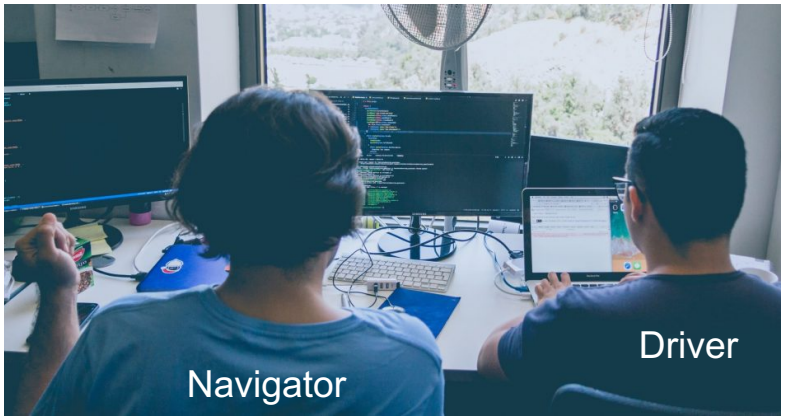
Recommended reading: A guide for reviewing code and having your code reviewed, by Thoughtbot
Reference: Expectations, outcomes, and challenges of modern code review, Alberto Bacchelli and Christian Bird

Pair Programming

- Two programmers work together at one machine.
- The *Driver* enters code and thinks tactically.
- The *Navigator* critiques the code and thinks strategically.
- Periodically switch roles and pairs.

Benefits:

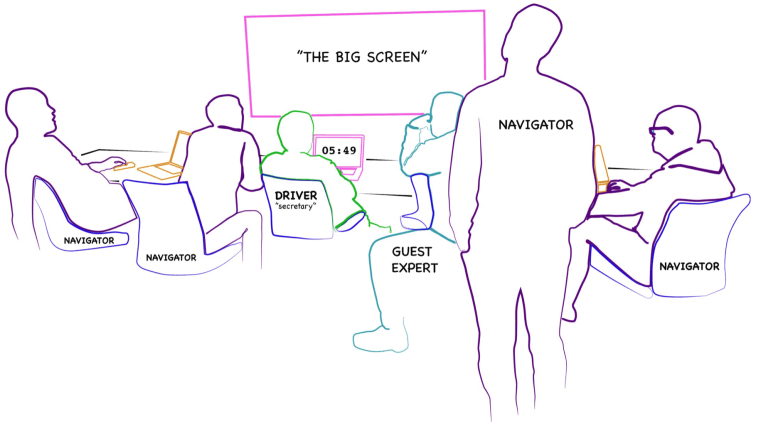
- Informal review process.
- Helps developing collective ownership and spread knowledge.
- Improves quality (less defects, better design) whilst maintaining (or improving) productivity.



Recommended reading: On Pair Programming, by Birgitta Böckeler.

Ensemble Programming
(aka Mob Programming)

- Basic rule: *"for an idea to go from your head into the computer it MUST go through someone else's hands."*
- The *Driver* sits at the keyboard and focuses on typing the code.
- The *Navigators* discuss the idea being coded and guide the Driver in creating the code.
- The *Driver* listens to and trusts the *Navigators*. The Navigators express their ideas to the Driver in a slow, metered approach.



- Each team member acts as the Driver for a short period of time (typically 10 to 15 minutes).
- To follow this approach we must become **good at communicating and discussing** each idea with another person before it can become part of the code base.

Recommended watching: Mob Programming and the Power of Flow, by Woody Zuill @ GOTO 2019

Continuous Integration

“What if engineers didn't hold on to modules for more than a moment? What if they made their (correct) change, and presto! everyone's computer instantly had that version of the module?

*You wouldn't ever have **integration hell**, because the system would always be integrated. You wouldn't need **code ownership**, because there wouldn't be any conflicts to worry about.*

[...] the fundamental assumption of CI is that there's only one interesting version, the current one”

Continuous Integration on the C2 wiki.

Should *feature branches* be used?

- They are useful to isolate and ensure stability of a shared code base.
- But **they work against continuous integration**! *Continuous Integration* (CI) implies integrating work at least daily, but probably multiple times per day.
- **Feature branches imply a cost** when the time comes to merge a set of changes. The longer-lived a branch is, the greater the cost might be!
- We can do ***trunk-based development*** by separating deployment from release. Some example techniques:
  - > Dark Launching
  - > Branch by Abstraction
  - > Feature flags

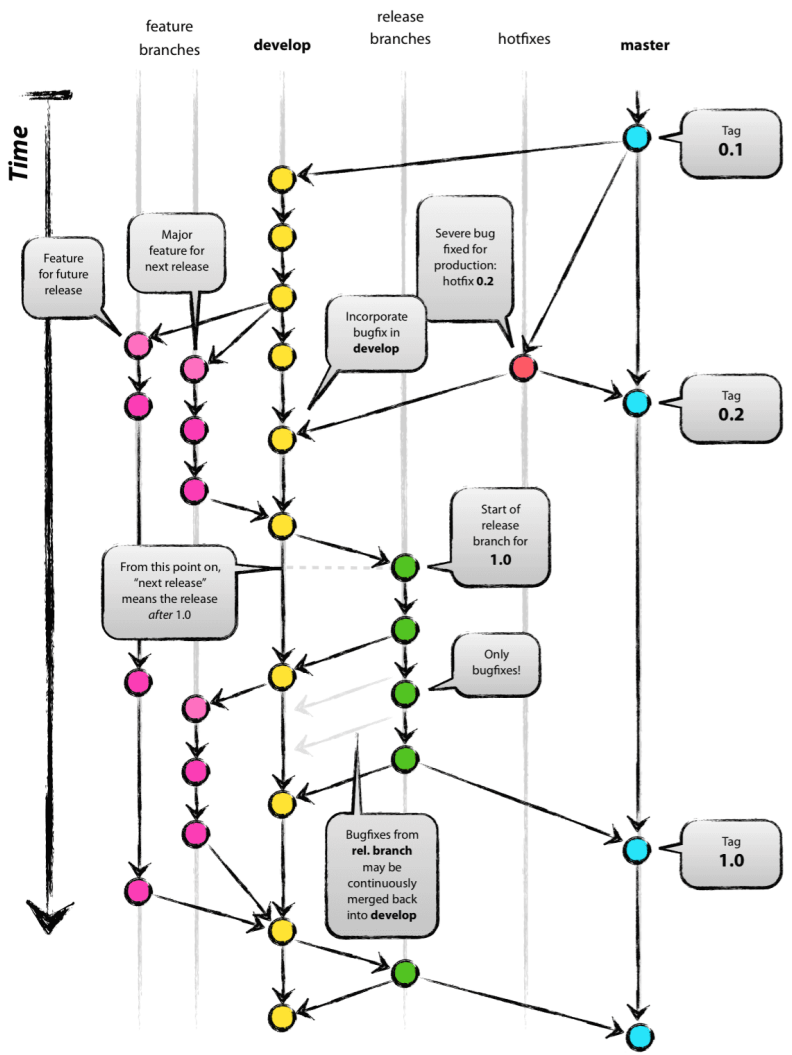
Recommended watching: *Continuous Integration vs Feature Branch Workflow*, by Dave Farley

Advice on Pull Requests

- Use small pull requests.
- Probably this means more than one pull request by backlog item.
- Each pull request should keep the source code stable.
- The code changes provided in the context of a pull request should be tested and documented within the same pull request.

What about GitFlow?

- It was never designed to be an all-or-nothing decision.
- You should pick the elements that make sense for your context.
- Some advice:
  - Avoid *develop* and *release* branches unless you really-really need them.
  - Have *feature* branches as short-lived as possible.



FEUP.DEI

DEI - DEPARTAMENTO DE ENGENHARIA INFORMÁTICA
dei.fe.up.pt | secdei@fe.up.pt | +351 225 082 134
Director **João MP Cardoso** | Full Professor
jmpc@fe.up.pt | +351 220 413 284

FEUP - FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO
Rua Dr. Roberto Frias | 4200-465 Porto | PORTUGAL
www.fe.up.pt | feup@fe.up.pt | +351 225 081 400
@DEI_FEUP

DEI DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

U.PORTO