GD2S02 – Software Engineering for Games

Code Review

Pair Programming

Overview

- Code Review
- Why Code Review?
- Types of code review processes
- Pair Programming
- Pair programming rules
- Why pair programming?
- Pair programming pitfalls

Code Review/Peer Review

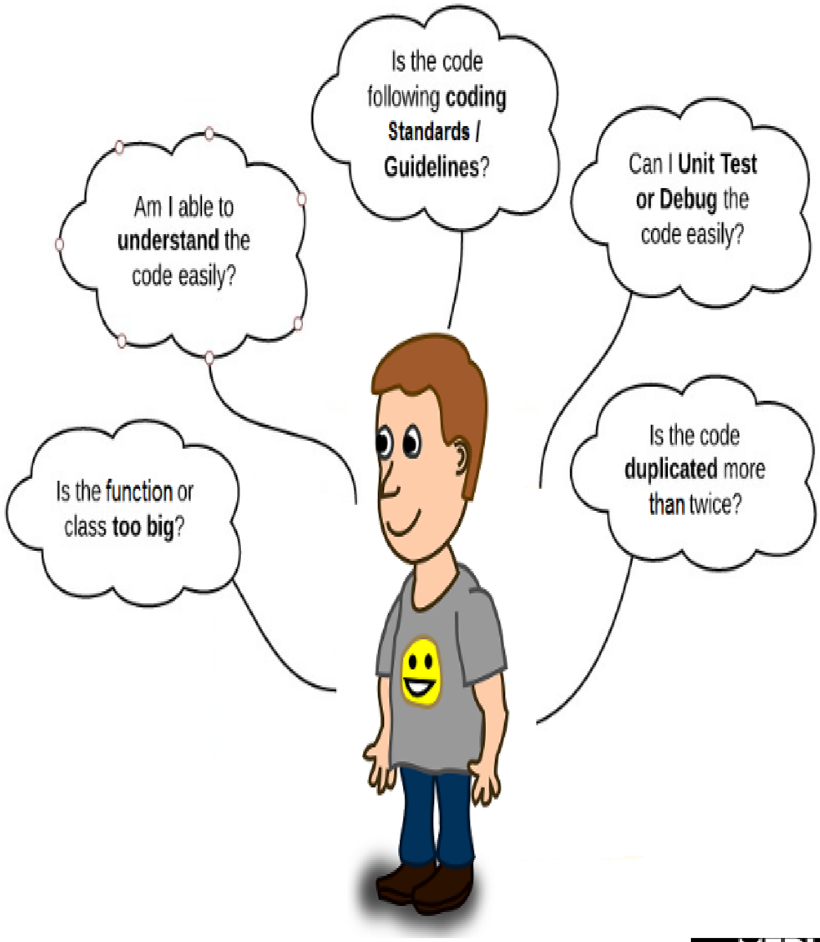
- Code review is a systematic examination of written source code, with the intention of finding mistakes that were overlooked in the initial development phase.
- The aim of code review is to improve the overall software quality.

Peer View

- “Peer review – an activity in which people other than the author of a software deliverable examine it for defects and improvement opportunities – is one of the most powerful software quality tools available. Peer review methods include inspections, walkthroughs, peer desk checks, and other similar activities. ” Karl Wiegers

What to look for in code review?

- Feature Completion
- Potential side effects
- Readability and maintenance
- Consistency
- Performance
- Exception Handling
- Simplicity
- Reuse of existing code
- Test cases covering all possible execution paths.


Is the code following coding Standards / Guidelines?
Am I able to understand the code easily?
Can I Unit Test or Debug the code easily?
Is the function or class too big?
Is the code duplicated more than twice?

MEDIA DESIGN SCHOOL GAME DEV

Why Code Review?

Code review helps you:

- Catch bugs.
- Ensure code is readable and maintainable.
- Spread knowledge of the code base throughout the team.
- Get new people up to speed with the ways of working.
- Expose everyone to different approaches.
- Finding and correcting errors at this stage is less expensive and tends to reduce the cost of handling, locating, and fixing bugs during later stages of development or after programs are delivered to users.

Types of code review

1- Formal Code Review (Inspections)

– Careful and detailed process
– It has multiple participants multiple phases
– A series of meetings is held to review code line by line.
– One of the participants is the "moderator", facilitator to keep everyone on task.
– Defects are recorded in great detail (location, severity, type,….).

Types of code review

2 – Over the shoulder review

- Informal code review
- A programmer standing over the author's workstation, while the author is walking him through the code.
- The author "drives" the review.
- Small changes could be done while reviewing, but big changes are taken off-line.
- Simple method, that lends itself to learning and sharing information between developers and improves communication.
- Since it is informal
  - Nothing guarantees that all code changes are reviewed.
  - No reporting or documenting.
  - The author can unintentionally miss a code change.
  - Normally the reviewer does not track whether the defect was fixed or not.

Types of code review

3- Email pass-around reviews

- Informal code review
- Source code files/changes are sent by the author to reviewers by e-mail. The reviewers then examine changes, ask questions, discuss changes and suggest possible modifications.
- Advantages :
  - Easy to implement
  - You can work with remote reviewers easily
  - Easy to bring more people into the review
  - Reviewers can choose the appropriate time to do the review without breaking their work flow.
- Disadvantages:
  - It can easily become difficult to track the reviews, even for trivial reviews, the mail threads can grow and become hard to track.
  - For remote review, it might take days due to time difference.
  - Still no way to make sure all code was reviewed.

Types of code review

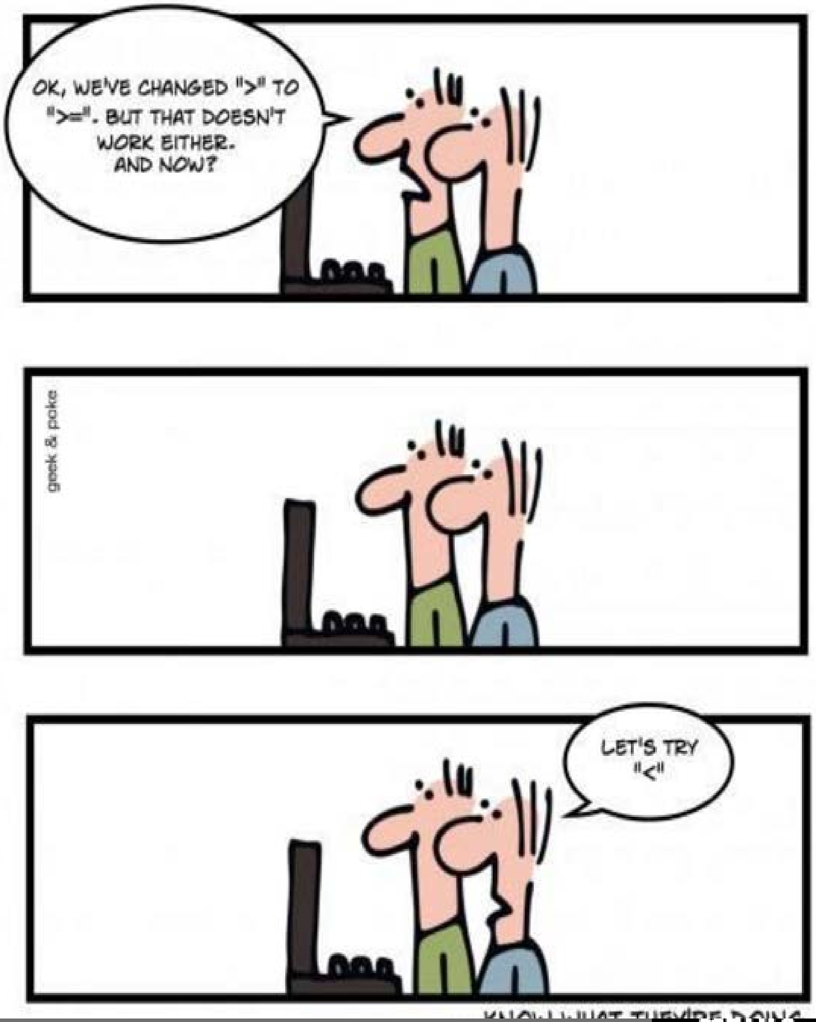
4- Tool Assisted Reviews

- A specialized tool is used to do the review.
- The tool is responsible for:
  - Collecting files
  - Transmitting and displaying files, commentary and defects among participants.
  - Collecting metrics.
  - Giving PMs control over the workflow.

Types of Code Review

5- Pair Programming

Pair programming consists of two programmers sharing a single workstation (one screen, keyboard and mouse among the pair). The programmer at the keyboard is usually called the "driver", the other, also actively involved in the programming task but focusing more on overall direction is the "navigator"; it is expected that the two programmers swap roles.


OK, WE'VE CHANGED ">" TO ">=". BUT THAT DOESN'T WORK EITHER. AND NOW?
geek & poke
LET'S TRY "<"

Pair Programming

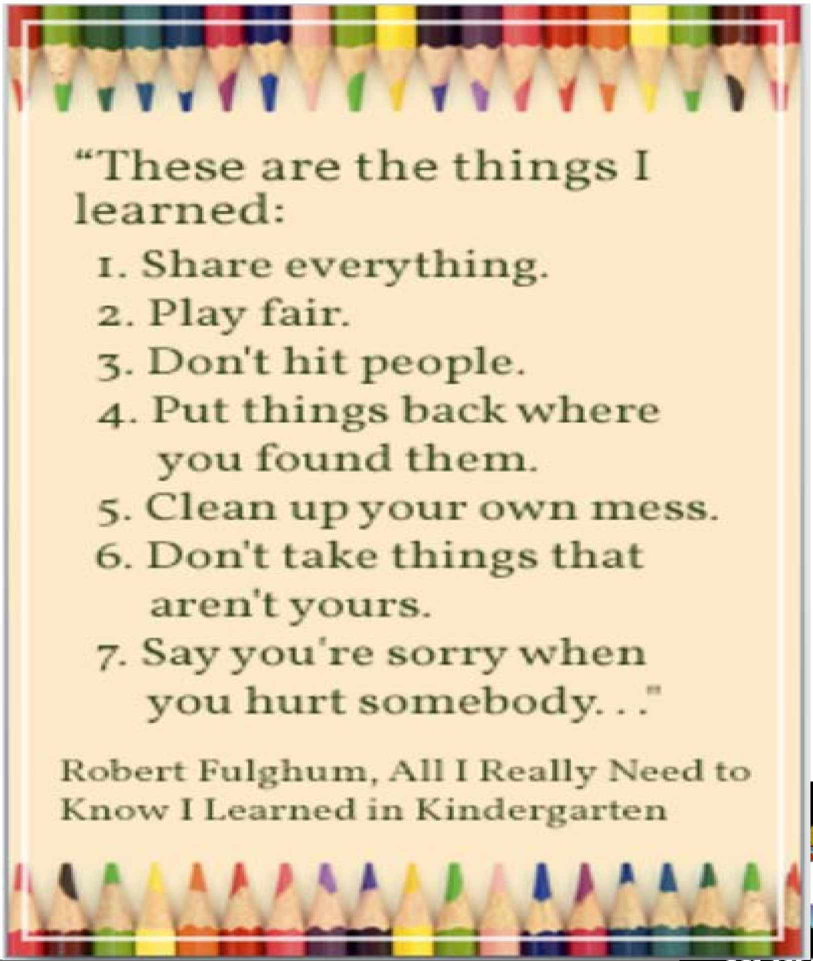
- Both programmers are continuously collaborating on the same design, problem, line of code, or test.
- While the driver types, the navigator observes, looking for tactical and strategic defects in the driver's work.
  - Tactical defects: syntax errors, typos, calling the wrong method, etc.
  - Strategic defects: defects in the implementation/d that might cause the implementation to fail.

Pair Programming

- Continuous conversation should always be going between the two developers. The driver should be speaking loud about what he is doing and the navigator should be speaking about the overall direction of the written code.
- Paired sessions can last between 2-4 hours a day. So programmers do not work in pairs all the time because it might be mentally exhausting.
- Pairs should rotate at frequent rate to avoid what we call *pair fatigue*.

Pair Programming Rules

- *"Share everything, Play fair, Don't hit people, Put things back where you found them, Clean up your own mess, Don't take things too seriously, Say you're sorry when you hurt somebody while moving the furniture, Wash your hands before you start, Flush, Warm cookies and cold milk are good for you, Live a balanced life, Take a nap every afternoon, Hold hands and stick together, and Be aware of wonder."* Robert Fulghum



"These are the things I learned:

1. Share everything.
2. Play fair.
3. Don't hit people.
4. Put things back where you found them.
5. Clean up your own mess.
6. Don't take things that aren't yours.
7. Say you're sorry when you hurt somebody. . ."

Robert Fulghum, All I Really Need to Know I Learned in Kindergarten

Pair Programming

- *Share everything: both programmers are responsible.*
- *Play fair: take turns driving, even if you are more experienced.*
- *Don’t hit people: but help your partner stay focused and involved.*
- *Put things back where you found them: be positive about yourself and your partner, put negative judgements in the trash.*
- *Clean up your own mess: having another programmer sitting by your side should help notice defects.*
- *Don’t take things too seriously: have he disagreement/debate about errors and defects.*

Pair Programming Rules

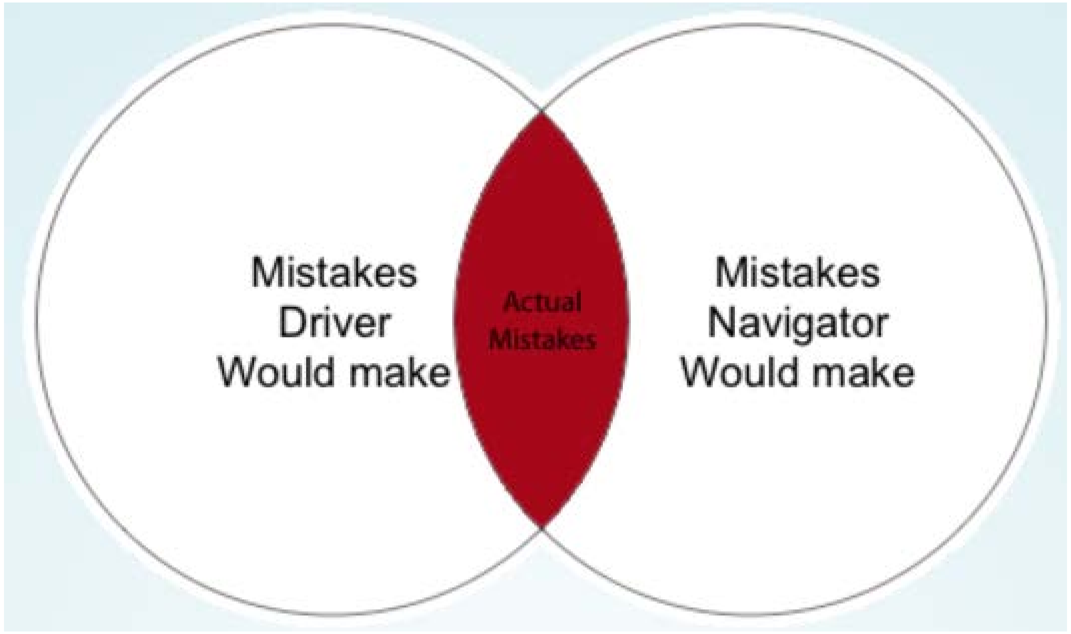
- *Say sorry when you hurt somebody while moving furniture: accept your partner opinion and when it is time to swap, slide the keyboard do not move the chairs.*
- *Wash your hands before you start: wash your hands from any doubts about your success.*
- *Flush: flush (discard) all the independent work that you have done on your own(if needed) or review it with your partner.*
- *Warm cookies and cold milk are good for you: take a break to maintain your stamina for another productive pair programming round.*

Pair Programming

- *Live a balanced life: communicate your thoughts with others, to allow for effective idea exchange and efficient transfer of information.*
- *Take a nap every afternoon: work separately, do experimental prototyping,….*
- *Hold hands and stick together: no competition between pairs, work cooperatively.*
- *Be aware of wonder: you will come up with more than twice as many possible solutions than if the two of you had worked alone*

Why Pair Programming?

- Better product quality
- Programs pass more tests than those written by individual programmers.



- Knowledge sharing, more than one person understands each area of the code. The project ends up with multiple people understanding each piece of the system.
- Better transfer of skill and diffusion of knowledge among the team.

Why Pair Programming?

- Code is more understandable.
- Force developers to follow proper programming practices.
- Code is continuously reviewed by the navigator, which increases the defect removal rates.
- Positive effect on productivity, although there might be initial decline in productivity when first introduced.
- People learn to communicate together and work together.

Why Pair Programming?

- More economical:
  - Two developers can usually solve a problem faster together than they would do alone, so coding is faster.
  - Increasing fault detection reduces the amount of time spent on rework and testing.
  - Pairs help each other to keep focused and on task.
- Pair programming is scalable, which gives both the developers and the system time to get used to it without being shocked.

Why Pair Programming?

- When two people have different specialties, their skills are transferred. Ad-hoc training occurs as one person shows the other some tricks, nice workarounds, ….
- Enjoyable for those who likes to work in pairs.

Pair Programming Pitfalls

- Both programmers must be actively engaging with the task , otherwise no benefit can be expected.
- Some people claims that that pairing "doubles costs“. This is a wrong assumption.
- Both programmers, are expected to keep up a running commentary, it is about “programming out loud“.
- Pair programming cannot be forced upon people.

Summary

- Code Review
- Why Code Review?
- Types of code review processes
- Pair Programming
- Pair programming rules
- Why pair programming?
- Pair programming pitfalls

References

- The Agile Samurai by Jonathan Rasmusson, The Pragmatic Programmers, 2010
- "All I Really Need to Know about Pair Programming I Learned In Kindergarten", Laurie A. Williams, Robert R. Kessler
- https://blog.codinghorror.com/pair-programming-vs-code-reviews/
- https://phinze.github.io/2013/12/08/pairing-vs-code-review.html
- https://www.codeproject.com/Articles/1156196/Code-Review-Checklist