# Informatics 1: Object Oriented Programming

## Tutorial 08

### Week 11: Code Review

Brian Mitchell (`brian.x.mitchell@ed.ac.uk`)
Vidminas Vizgirda (`s1750767@ed.ac.uk`)

## 1 Introduction

This tutorial is centred around improving your Assignment 3 Part 1 submission, so that your Part 2 is likely to be higher quality. It also concerns making more use of your tools. You will use IntelliJ to help identify and solve problems.

There is an understandable temptation to go straight to Part 2. However any errors or flawed design unknowingly transferred from Part 1 will exacerbate the complexity that Part 2 introduces.

If you have not submitted anything for Assignment 3 Part 1, you will not have individual tutor feedback, but you can still work through the IntelliJ code review with what you have so far. If you have not started Assignment 3 at all yet, then you can use this tutorial as a chance to start now.

In 1977, a previous incarnation of Edinburgh's School of Informatics updated its own multi-user operating system – the Edinburgh Multiple Access System (EMAS. Report PDF). Uptake was limited as it was tied to a particular type of computer. The released version was an acclaimed reworking of their first attempt at writing an operating system: a task large and complex even for experienced computer scientists. The quality of the end product was driven by the philosophy of the redesign:

> Our problem is that we never do the same thing again. We get a lot of experience on our first simple system, and when it comes to doing the same thing again with better [equipment and knowledge], we try and produce something which is ten times more complicated and fall into exactly the same trap. We do not stabilize on something nice and simple and say **"let's do it again, but do it very well this time."**
>
> – David Howard (in Hoare, CAR and Perrott, Ronald H, eds., Operating Systems Techniques, Academic Press, 1972).

The main task for your tutorial is improving the quality of your code. Your tutor will do a brief code review with you and you will self-study a longer review using IntelliJ's code review tool.

## 2 Let's do it again, but do it very well this time

### Task 1 - Preparation ◁ Task

It is strongly advisable to leave your Part 1 submission unchanged for reference so that you can see how far you have come, or if you need to abandon new changes and return to the older version. You can either make a copy of your Part 1 IntelliJ project or use version control (if you have not used it yet, some help materials are available in the bonus self-study tutorial this week).

Once your progress version of Part 1 is ready, open the Problems tab, then run a code review: Code » Inspect Code. . . » Whole project.

The inspection results are shown in the Problems tab. Exactly what is shown depends on your code. One key category to note is `Probable bugs` but you should look through the entire list. Some of the findings may be because your code is unfinished, for example `Unused declaration`.

Some common Java rookie errors at this stage of your progress are often related to misusing `static`:

**Instantiation of utility class**
A utility class is `static` so it does not and should not need instantiating (creating a variable of the type of that class).

**Access static members via instance reference**
An example helps. To turn a number into its `String` equivalent (the value 87 to the `String` `"87"`) the `String` class provides a static method (function) called `valueOf`. A rookie error is to think you need an existing `String` in order to call `valueOf`:

```
// This is unnecessary
String useless = "";
String useless87 = useless.valueOf(87);

// This is even worse because the value of the variable "misleading" does not matter
String misleading = "1";
String misleading87 = misleading.valueOf(87);

// This is correct
String s87 = String.valueOf(87);
```

## Task 2 - Tutor code review (5 minutes per student)

◁ **Task**

At some point during the tutorial, your tutor will talk to you about your Part 1 submission. To maximise this precious time, please have the submitted version of your Part 1 quickly available in IntelliJ with the `Inspect Code` already run and displayed. If you have made significant progress in your revised version then you might prefer to discuss the newer version.

## Task 3 - IntelliJ code review

◁ **Task**

While you wait for a tutor to come and discuss individual feedback with you, work through the suggestions IntelliJ provides in the sub/tab generated by the code analysis. When you click on an item, IntelliJ will show you the relevant code snippet and above it will be a light bulb similar to 💡 for context-sensitive changes.

You can also see IntelliJ's suggestions in the code editing window by using Ctrl+Alt+↑ and Ctrl+Alt+↓ to go to the next and previous problems respectively.

It is possible that IntelliJ might not have any recommendations for you. If that is the case, use this opportunity to revise the Code Quality document and makes notes to yourself

Some IntelliJ tips and tricks:

- As well as helping fix errors, IntelliJ's context-sensitive actions lets you alter code easily into something functionally equivalent. This allows you to experiment to find the most readable version of your code. The context-sensitivity depends on **exactly** where the cursor is.

  - If you place it on the keyword `for` and press the shortcut key for context-sensitive actions Alt+Enter then one of the options will be to unroll the loop which then shows you all the individual steps that loop takes. Conversion between `for` and `while` is available.
  - Actions for some `if` statements convert between `if` and `switch`.
  - Placing the cursor on the boolean operators such as `||` lets you alter the ordering or invert the conditions.

- Alt+Insert is another way to insert automatically generated code such as getters, setters, and constructors. Getters and setters are discussed in the Code Quality and Conventions document accompanying this course's assignments. That document also lists the conventional ordering of components in a Java class. However you do not need to remember the ordering: all you need to remember is that IntelliJ has Code › Rearrange Code to do it for you.
- ⇧+F6 (⇧ is the shift key) lets you rename an identifier everywhere it is used.
- Ctrl+Alt+L reformats code (all of it or just the selected bits).
- Ctrl+W can be used to widen the selection (add ⇧ to shrink).
- All these shortcuts are for commands that are also available in the Edit, Navigate, Code, and Refactor menus. If you expand a menu, shortcut keys will be displayed next to each command. Memorising ones that you use frequently can save you a lot of time!

## Task 4 - Reflection (last 10 minutes)

◁ **Task**

It is well worth spending the last part of the tutorial comparing your revised version with your original. Note what you have changed and think about how this counts as improvement. Don't judge improvement solely by the number of problems removed but by the new knowledge you have acquired and existing knowledge you have reinforced. Readability of your code is also extremely valuable because in a professional environment code is read more often than it is written.

There is no structure for this part, you've already practiced writing structured reflection. Feel free to write your reflections down in whatever format you wish. We do recommend writing reflections down, as it will help you to remember them.

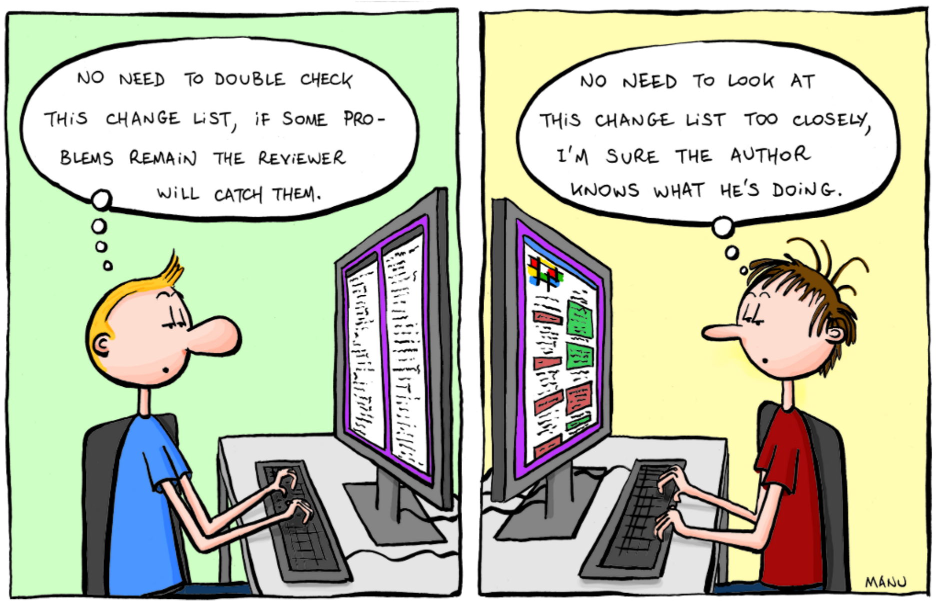
NO NEED TO DOUBLE CHECK THIS CHANGE LIST, IF SOME PROBLEMS REMAIN THE REVIEWER WILL CATCH THEM.
NO NEED TO LOOK AT THIS CHANGE LIST TOO CLOSELY, I'M SURE THE AUTHOR KNOWS WHAT HE'S DOING.
MANU

Figure 1: “Code Reviews” by Manu Cornet is available under the CC BY-NC-ND licence.

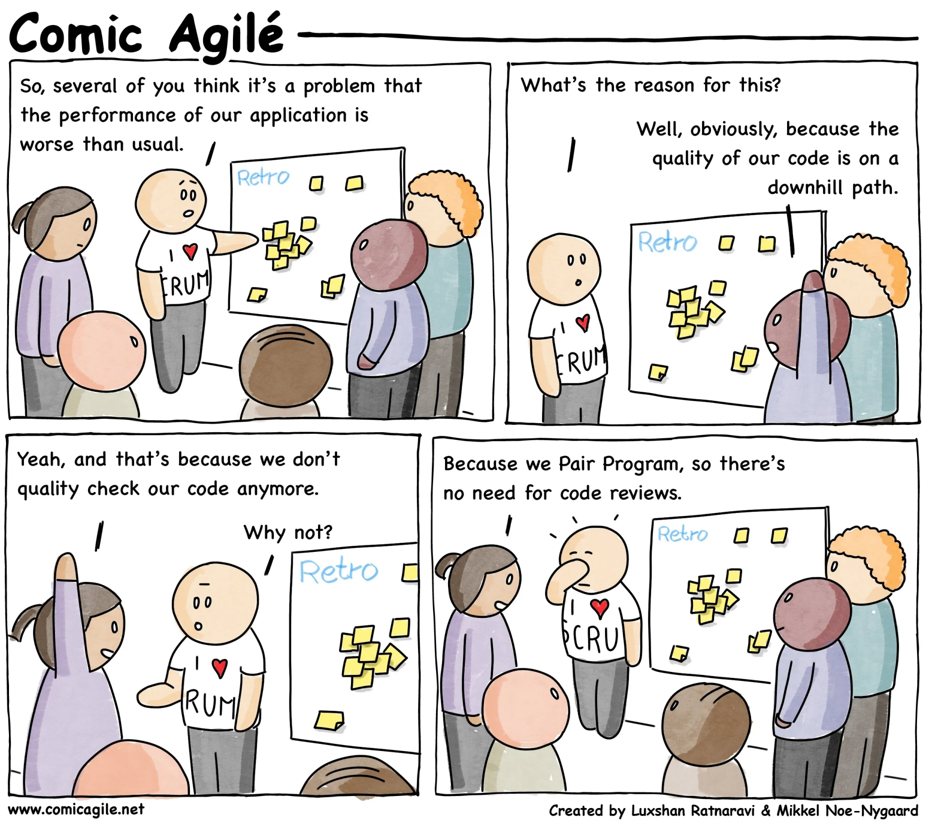
Comic Agilé
So, several of you think it's a problem that the performance of our application is worse than usual.
Retro
I ♥ SCRUM
What's the reason for this?
Well, obviously, because the quality of our code is on a downhill path.
Yeah, and that's because we don't quality check our code anymore.
Why not?
Because we Pair Program, so there's no need for code reviews.
www.comicagile.net
Created by Luxshan Ratnaravi & Mikkel Noe-Nygaard

Figure 2: “#136 - Code Quality” by Comic Agilé (Luxshan Ratnaravi & Mikkel Noe-Nygaard) is available under the CC BY-ND licence.