**Anti-Pattern:** The Blob.

**Project:** PROG8170 Software Quality Assurance Techniques, Assignment 2

**Details:** This project required several versions of a chat app to be made, one of which was a desktop version. One feature it included was the ability to load history, so after you entered your user id you could see previous messages for yourself (or public messages).

However, as the work on the project continued, it was realized that connecting to the database, browsing the message history, checking if messages were for a particular user, making sure messages were valid and did not contain any corrupt data, among other things, were a lot of different operations and the method that loaded history had, in fact, evolved into an example of “The Blob" anti-pattern. It was a typical case of it being hard to foresee exactly all the details completed to do something until testing was well underway.

It was broken up into several smaller methods after the peer-to-peer testing was done (the instructor for this course required that within each team, everyone's code was tested by another team member and the results documented).

**Lessons learned to avoid it happening again:** Whenever you are not fixing problems by changing existing code, but adding new code, ask this question: does this belong in an existing module or method, or is its purpose to solve one very specific, focused problem? If so, should it be in an individual, cohesive module or method?

1. **Anti-Pattern**: Functional Decomposition.

**Project:** For PROG8180 Web Technologies, one assignment required the creation of a Concentration card game.

**Details:** This assignment involved displaying a set of playing cards, face down, and allowing the user to look for matching pairs. This meant that for each card it was necessary to always keep track of its current state, face up or down, and to turn it face down again if it was not matched with another card, but to keep it face up when it had been matched. It also required each card to have properties describing what specific card it was.

This was done by keeping arrays of flags and data to keep track of all these properties. This was, however, more like old style procedural based programming rather than proper modern, object-oriented programming. A better solution would be to treat each card as an object, with properties describing its position on the board, its state (face up or down), whether it was matched yet, and anything else needed.

**Lessons learned to avoid it happening again:** When dealing with newer languages that a developer is unfamiliar with, it can feel overwhelming trying to learn its nuances and apply them, and it is easy to fall back on more familiar techniques if they do work and one knows them well enough to complete some work quickly. In order to cope with that and take advantage of the power of new languages and avoid that feeling of overwhelm, it would help not to focus so much on trying to solve the whole problem, but break it down into small parts - even just making a simple object, with one or two properties one needs and making sure that this object works, then adding more might seem like slow progress at first but would result in the kind of better, more maintainable code that can save time in the long run.

**Anti-Pattern:** Walking through a Minefield

**Project:** PROG8180 Web Technologies, Assignment 2

**Details:**CakePHP was to be used with Composer. However, neither the instructor nor the students were familiar with these. Although some people had used them in the past, it turned out that the current version did not work well – the combination of CakPHP and Composer was not good, and the instructor needed time to learn how to use CakePHP on Windows since he was more familiar with Mac. The result was that a number of students and the instructor spent a lot of time trying to make a system work that was not very feasible to use, and eventually the class switched to a different approach, using CakePHP without Composer.

**Lessons learned to avoid it happening again:** In a situation with time constraints, like trying to get an assignment done by a deadline, it is best to verify that the software to be used will work on the platform to be used before rolling work out to a large group, i.e. students in a class or a project team in a company. Find something that at least basically works well before asking people to build on it extensively.

**Anti-Pattern:** Input Kludge

**Project:** PROG8180 Web Technologies, Assignment 1

**Details:** This assignment required the creation of a pizza ordering application that allowed a user to enter personal information and details of what kind of pizza they wanted, toppings, etc. However, problems could arise when trying to complete an order if the input was not good, i.e. postal codes or telephone numbers did not follow the correct formats, incomplete data, etc.

**Lessons learned to avoid it happening again:** Validating data carefully often does take time but is worth the effort. Some aspects of how to solve this problem are readily available. In this case, it was not difficult to find online examples of regular expressions that could be used in JavaScript to validate things like Canadian postal codes, phone numbers. Using these solutions works well with being thorough, making sure all necessary data is collected and validated and test a variety of combinations of input.

**Anti-Pattern:** Cut-And-Paste Programming

**Project:** INFO8250, Mobile Application Design Assignment 1, Widgets

**Details:** One part of this assignment required the implementation of three SeekBars for an Android App. This meant that the logic for each was quite similar, so it seemed easy to save time but copying and pasting code. However, many small details of the code had to be made unique for each SeekBar object, which was time consuming to fix and created a few difficult to spot bugs.

**Lessons learned to avoid it happening again:** When using an Integrated Development Environment like Android Studio and using multiple occurrences of the same type of object, it is common for each to require its own listener to be implemented. One solution worth considering is making each listener send a common set of parameters to an interface that then is generic and able to handle any one of that type of object’s needs, perhaps by creating a separate class with methods that can be used. This may take more time to set up, but could save time later when maintaining code.

**Anti-Pattern** Lava Flow

Dead Code.

Most projects are based on legacy project.

But most project have no reflection or refactoring the project codes, because most

company don't want spend on time that is not productive, although it is productive in the point of overall view at the end.

So, some of the codes are added during the end of phase of project for fixing some bug that is the temporary solutions, not right way to solve the problem. However the software engineer that made code will be involved at another project when he finished project. So the opportunity to make code clean disappeared and new project will start based on dirty code. Moreover, these temporary solution make problem at the field that is not filtered out during QA process.

**Lessons learned to avoid it happening again:** To minimize that problem at Capstone project, TDD development can be used. If we make code testable, the code structure will be structured and test code can be a secure net for refactoring.

**Anti-Pattern** : Golden Hammer

One of mistake in industry is once tool is adopted, then it is hard to change, because it costs for educating employee and changing infrastructure. In 2000, my previous company adopted Test Director made by Mercury (Now it is part of HP) for managing Issues. It is one of best issue management tool and it is also expensive. Once adopted, it is hard to change due to accumulated data and costs for learning for new tool, although there are lots of cheaper and productive tools. Thus, QA part that is part of production division use Test Director and Research Lab use Jira for managing issues and Software Developers have to see both systems. Last year, new project for copying bugs from Test director bug into Jira system to lessen the burden that software engineer 2 issue tracking systems.

**Lessons learned to avoid it happening again:** When we select tools or technologies for Capstone project, it should be decided considering our project not popularity.

**Anti-Pattern** : Cut-and-Paste Programming

Nowadays most programmers use google and stack overflow for their coding. Most project in the industry are started based on legacy project. Thus, most engineers are accustomed to use merging tools such as beyond compare. However, as project is going on and going through QA process , codes becomes Spaghetti codes, because they are merged not considering overall structure. One of the reason might be most cut and paste code are made by Junior programmer, and Senior programmer does not have time or company have no process for peer reviewing.

**Lessons learned to avoid it happening again:** For student cut and paste programming is unavoidable, so peer reviewing can be conducted to avoid unnecessary code injected during the Capstone project.