Duplications in the Context of Software Maintainability

dr. Adam Vanya

INTRODUCTION

Duplicating source code is a debatable action developers do during software development. There are several reasons why developers are actually performing it in spite of all the hurdles duplicates cause for software maintainability. This blog post describes the topic of source code duplication, the effects of the duplicates on maintainability, the reasons for duplicating and the way source code duplication can be handled.

DUPLICATING

Every now and then, developers create new code segments which are completely, structurally or just functionally the same as other, already existing portions of the code base. This process is often referred to as code duplication or code cloning. The resulting related code segments are the so-called duplicates or clones. It is possible that there are more than just two duplicates related to one another: developers can duplicate a given code segment as many times as they want.

Furthermore, code duplication can be intentional or accidental. When a developer selects, copies and pastes a piece of code, then it is done with an explicit intension. However, a newly written code portion can be a duplicate just because a developer happened to write the same lines, the same code structure or the same functionality that already exists somewhere else in the code base.

Identifying and tracking which portions of the code base are duplicates of one another is an extremely difficult and an effort intensive task without the proper software tools. The complexity of identifying duplications comes from the sheer size of the code base and the fact that each portion of the code base has to be compared to another portion to detect duplicates. Unlike static calls, code duplicates do not expose visible clues about the related code fragments.

Also, software changes continuously as a result of its changing environment. In the new versions of a code base, duplicates may be added or removed. Even if a developer would remember about all the duplicates he or she introduced, we need to face the fact that a code base is typically maintained by several developers and they need to understand, extend and modify the code written by their colleagues. Therefore, it is not surprising that developers often do not have a reliable picture about the existence and location of all the duplicates.

THE HARM DUPLICATION CAUSES FOR MAINTAINABILITY

There are several ways how duplications can hinder the smooth maintainability of software systems. One of them is related to the size of the code base. Each duplicate increases the size of the code base much more than a code which simply calls an already existing code fragment or reuses it with another technique, like inheritance. Duplication therefore leads to a certain amount of unnecessary, redundant code which also needs to be maintained. Since more code is more difficult to analyze and test, duplication negatively affects maintainability. Each instance of duplicated code needs to be tested as well. Therefore, duplication also leads to more tests which are partially or completely redundant.

Creating duplicates is easily done by copy-pasting a piece of code. However, if the original code contains a bug then such a bug gets duplicated too and it needs to be fixed not only once, but multiple times. The more copies the developers creates of the buggy code, the more instances of that bug needs to be fixed. Fixing issues multiple times have a significant impact on the effort spent on maintenance.

As discussed earlier, developers have only a limited amount of knowledge about which code segments are duplicated and where exactly the related duplicates are located. Because of this, developers do not always change all the duplicates which belong together. Also in such cases, they assume that their changes are complete but in fact they introduced inconsistent modifications to the code base. Understanding why the same functionality is implemented many times and in different ways makes the understanding of the source code more difficult and therefore less easy to maintain. An inconsistent change may further result in unexpected behavior, crashes and a false believe that a bug is completely fixed. These new and remaining issues need also to be handled and they take up extra maintenance effort and therefore costs.

Related duplicates from different components increase the coupling between those components. The stronger components are coupled, the less likely modifications can be kept within the boundaries of a single component. When the execution of development tasks involve the modifications of multiple components then maintenance costs increase due to the extra development, testing and communication efforts.

By introducing duplicates, developers explicitly or implicitly avoid the effort that is needed to make a piece of code reusable and therefore more maintainable. For instance, a code fragment is not extracted into a callable method or a proper inheritance structure is not set up. Refactoring source code for better reusability costs, of course, some effort but one has to know whether or not it exceeds the effort spent due to the presence of current and future duplications. Creating duplicates costs effort as well. The more duplicates of a code fragment we create the higher the chance is that refactoring the code to be reusable at the first place would have been cheaper.

Duplicating code a lot may also be an indication about the lack of knowledge developers possess about maintainability and its importance. If developers are less aware about this topic then they are likely to violate other maintainability related rules as well. Consequently, they will keep creating poorly maintainable code, unless they receive the proper education and management instructions. The fact that developers are able to check-in source code with duplicates created may further be an indication of the shortcomings of the development process applied.

The extent of the harm duplications can cause for software maintainability depends mostly on the number of times a piece of code got duplicated, the size and complexity of the duplicated code, the number of components the duplicates reside in and the number of developers involved in the creation of the duplicates of the same code fragment. These factors can also be used to select which duplicates should be refactored first for reaching the greatest positive impact on maintainability.

REASONS WHY DEVELOPERS DUPLICATE

The main goal of developers is to execute the tasks they got assigned to. These can be roughly feature implementation, feature enhancement and bug fix type of tasks. The execution of these tasks cannot take forever: developers are pushed to get ready as soon as possible and pick up their next task assigned. Within such a stressful context, it is understandable that even experienced developers take shortcuts. Duplicating a piece of code without refactoring the code to be reusable is simply quicker and requires less thinking.

In many situations, when the code belongs to another development team simply duplicating the code also reduces the amount of communication needed between the different teams to properly introduce the needed interface changes. Communication between development teams not only takes time but it needs extra planning and coordination.

Some developers do not have the proper knowledge about maintainability and the effects of code duplications. In such cases, developers need to be educated so that they can knowingly decide about when duplicating a piece of code is really the only reasonable alternative.

The lack of guidelines, measurements and acceptation criteria during a software development process and personal evaluations may result in a work environment where developers do not need to face the consequences of their choices about duplications. Therefore, it is relevant to mature the development processes and personal evaluations in such cases.

Another logical explanation for developers to duplicate source code is to be able to defend themselves. If a piece of code was accepted and it is already a part of the code base then developers may feel safe to use this as an argument and copy that code. Something similar happens also when developers copy code snippets from an internet forum. Such behavior is risky and the development process is responsible to make developers understood that blindly duplicating without proper argumentation is not acceptable.

HANDLING DUPLICATONS

Just as handling any problem, handling duplications should start with a fact-based analysis of a given situation. With proper measurement tools, the location, size, complexity and the authors of duplicated code can be identified. The data gathered can then be used to prioritize the duplications based on the harm they may cause for maintainability. It seems to be wise to fix the most harmful duplications first.

Fixing may involve creating callable methods or functions, developing inheritance structures and creating or modifying interfaces between components.

The above describe approach to handle duplicates is reactive. It is possible and advisable to combine this approach with proactive approaches in order to avoid or minimize the appearance of duplicates in the code. Educating developers on maintainability, continuously measuring the duplications in the code, creating and applying a software quality centric software development process and personal evaluations are great ways to prevent duplications to appear and negatively affect software maintenance.

SUMMARY AND CONCLUSIONS

One way or another, software maintainability is affecting you. It can have an significant impact on your carrier if you are working for a software development company and it is a major factor to consider when you invest in such companies or buy custom software. Therefore, we believe maintainability has to be understood and explicitly be controlled to reach your goals.

In this blog post, we gave a brief description about source code duplication. We also reviewed the effects duplications have on software maintainability. We showed that duplications do have a significant and widespread negative impact. Furthermore, we detailed some of the possible reasons why developers create duplicated in spite of their disadvantages. Finally we provided concrete suggestions to handle source code duplications.

In case there are certain statements in this blog post you do not or not completely agree with or if you think it should be extended in some ways, please feel free to contact us and tell us about your idea: adam.vanya@frontendart.com