# The Effects of Software Maintainability

dr. Adam Vanya

**INTRODUCTION**

In our previous blog post titled “Factors Influencing Software Maintainability”, we analyzed the main factors having an impact on software maintainability. It helped us enumerate those things that we can monitor and change to get in control. In order to properly understand the importance of such control, we need to know what and how the level of software maintainability affects. This is the topic we are about to explore in this blog post.

**SOFTWARE MAINTAINABILITY EFFECTS**

The first and most obvious consequence of poor software maintainability is the increased amount of time and effort spent on the development activities (e.g. bug fixing, feature implementation). This decreases the envisioned profit and increases the chance of missing important deadlines. These missed deadlines can then contribute to losing in a market competition and ending up with a shrunken market share.

Software systems that cannot efficiently be maintained because of their source code properties will likely perform worse on other software quality aspects as well. For instance, source code with many complex and difficult to understand constructions will lead to a higher number of bugs in the development and production codes. This leads to an increased risk of executing bugs in production time causing software failures. Each failure is a potential threat to the availability of the software product or a service which is based on that product. In other words, software and service reliability can both be affected.

Another example touches upon software security. Imagine that a piece of code is duplicated many times and contains a serious security bug allowing hackers to steal valuable information. Even if this security breach is detected on time, because of the many copies of the code segment, there is an increased risk that not all duplicates will be modified. In such a case, hackers will still have an opportunity to circumvent security and break into the system.

As we can see, software maintainability has an important and quite distinctive supporting role regarding other software quality attributes. When we look at this the other way round, we can observe that the reasons for stakeholders to be unsatisfied can often be traced back to maintainability issues. Since letting down stakeholders can cause serious problems (e.g. losing market share or financial support), it is clear that software maintainability cannot be neglected.

The environment of most software changes constantly. This poses a never ending need to change the software so that it can work according to the new expectations. To be responsive and implement all these

changes, the deadlines have to be kept tight. If the maintainability level of the source code is low then developers will have a harder time meeting those strict deadlines, their stress level will increase and they will focus more on implementing features while taking less care about creating maintainable code. Under such working conditions, developers become more frustrated, less motivated and it is also more likely that they will consider quitting their job. Furthermore, it is more challenging to find new developers who are willing to work under those circumstances. Even if companies succeeded in increasing the number of employed developers, that would mean more labor costs and a short term solution that does not address the root cause of the problem. To read more about this topic, please refer to [1].

As for the developers, poorly maintainable code affects many aspects of their everyday job. Planning is one such aspect. Planning should be as accurate and as quick as possible. That is one of the reasons why they play planning poker in agile projects. However, if the code is difficult to understand and to analyze then evaluating what should be changed and which impact those changes would have, takes more time. Also, the accuracy of the resulting plans is negatively affected in such cases.

After planning, developers start to execute the development tasks assigned to them. They will introduce new bugs during the development activities more likely if the software system they work on is more difficult to maintain. One reason for this is that complex code gives more room for incorrect interpretations of the source code to be modified. The newly introduced bugs have to be fixed as well. The extra effort spent hinders developers to grow professionally as fast as they could have. It is no developer's dream to keep fixing bugs and doing so makes them lose internal motivation.

Developers like to be productive and handle as many challenges as possible in a short amount of time. Unfortunately, when the source code is hard to maintain then they need to spend more time on their development tasks to analyze the problem, introduce the changes needed and test the code developed. This phenomenon makes the development work monotone and causes demotivation.

## THE VICIOUS CYCLES OF MAINTAINABILITY

You might already have noticed that there is a factor causing maintainability to degrade while being also present as a possible consequence of degraded maintainability. This factor is the state of the source code as expressed by the maintainability relevant properties. Unless explicit effort is done to preserve or improve these properties, they will become worse after each maintenance cycle and so will the level of maintainability as a consequence. Studies show, see for instance [2], that this degradation is not even linear, but exponential in time!

When the maintainability is getting exponentially worse, all the negative consequences described in the previous section are getting magnified as well. Those software development companies which are not monitoring and managing the maintainability related properties of the source code can quickly find themselves in a situation where adding new features to the software and fixing bugs are extremely hard and takes an amount of time that is difficult to explain. This is the point in time where the software reaches its legacy state.

The development of legacy software is typically stopped and a new system is built to replace the existing functionalities. This always costs a tremendous amount of resources. How close a software system is to

become legacy cannot be known exactly without executing the necessary investigations and measurements. It can only be felt by the amount of pain during the maintenance process. The vicious cycle described here is strongly related to the vicious cycle of technical debt as discussed in [3].

## SUMMARY AND CONCLUSIONS

One way or another, software maintainability is affecting you. It can have a significant impact on your career 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 be controlled explicitly to reach your goals.

To help you understand the importance of such control, we went through the possible and relevant effects of software maintainability. We also described the vicious cycle of maintainability that is responsible for the acceleration of the maintainability-level degradation and the exponentially growing magnitude of negative consequences.

The next blog post in our series on software maintainability will explain what type of measurements are needed to get a good understanding about the current state of software maintainability. With these blog posts, we hope to increase awareness about software maintainability, help you control this concept and start constructive discussions about it.

In case there are certain things described 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

## REFERENCES

[1] Brooks, Frederick P., Jr., 1931-. (1982). The Mythical man-month : essays on software engineering. Reading, Mass. :Addison-Wesley Pub. Co.

[2] Bakota, T., Hegedűs, P., Ladányi, G., Körtvélyesi, P., Ferenc, R., & Gyimóthy, T. (2012). A cost model based on software maintainability. 2012 28th IEEE International Conference on Software Maintenance (ICSM), 316–325.

[3] Besker, T., Martini, A., & Bosch, J. (2018, May). Technical debt cripples software developer productivity: a longitudinal study on developers' daily software development work. In *Proceedings of the 2018 International Conference on Technical Debt* (pp. 105-114).