# Personal Reading: Fault Prediction

Mats Stijlaart - University of Amsterdam, The Netherlands

February 21, 2015

## What is Fault Prediction?

Fault prediction is a process to predict parts of a piece of software that are likely to contain a high amount of defects. With the use of tooling you can process source code and the tooling will address the components that may contain the defects. The methods used to perform the fault detection are treated in the section *Forms of Fault Prediction*. This process helps understanding the required maintenance effort for testing and resource allocation [4, p. 415].

The tooling gives an approximation of the parts of the software that are likely to have more faults. Based on this result, the software team can decide on which parts require more focus or which components require more resources [4, pp. 415-416]. To make this decision one must considerate the importance of these software components in the totality of the code base.

The main benefit of fault prediction is the ability to mitigate risks. The amount of defects can be reduced in an early stage, making it less likely to have failing software. On the other hand, there is a required investment to actually apply the risk mitigation. Knowing what parts of the software may contain a lot of defects does not change anything. A team must still actively reduce the risk.

## Forms of Fault Prediction

There are different types of fault prediction. Catal et al. classifies different studies in four categories [3]. The categories in which he adresses these studies are:

- The type of metrics. This implies if the metrics are applied on method, class, file, component or process level.
- The applied method. If statistics, machine learning and/or an experts opinion is applied. These methods could be combined.
- When the research was applied. This classification contributed to determining if the focus on method and/or metrics are shifting.
- The kind of dataset that was used. A dataset can be private or public or partially public. This identifies if studies are reproducible.

This classification of Catal et al. gives an overview of the performed research on the topic. However, I find that Catal et al. make some insubstantial conclusions. For example, he concludes that there is less motivation to use statistically based methods but instead machine learning [3, p. 7350] because machine learning is more commonly used in the newer literature. That is a too hasty conclusion in my opinion., since the capabilities of researchers increased over the last years. Hardware improved, publicly shared data sets were made easier due to companies as GitHub[1] and machine learning is easier to apply due to products as *Tensor Flow* [1]. These developments may have made these options feasible. Finally, Catal et al. state: "as specified in this review, machine learning models have better features than statistical methods or expert opinion based approaches" [3, p. 7351]. That is a quite insubstantial statement to make, on which the

[1] https://github.com

authors moreover fail to elaborate and for which they give no arguments whatsoever in the paper.

Another example is a claim that "we need to increase the percentage of papers using public datasets and 80% can be an ideal level" [3, p. 7351]. First there is no insubstantial argument for the '80%'. I agree that public data sets increase the reproducibility of the research, which should be a goal in research. However, private data sets may be more representative since they might be significantly bigger than public software projects as they probably include years of development. The point made, we cannot set a limit for 80%, but we have to look for representing data that improve the field.

The authors of *Making Software* [4] choose a different approach. In this book they focus only on metrics, but classifies these metrics in six categories. These are Code Coverage, Code Churn, Code Complexity, Code Dependencies, People & Organizational Metrics and Integrated/Combined approach. The results of this study show that Organizational Structure measures show the highest accuracy on their case [4, p. 430]. This result is explainable when you realize that when a lot of different minds work on a single piece of software the ideas may start to differentiate, which may lead to more bugs. However, when you turn the idea around, the reason why different people work on the same piece of software may come from different requirements that may contradict or interfere with each other. The point I want to make is that the arise of these bugs may not be directly associated with different developers working together, but might originate from another part of the organization.
The discussed research seems to focus on finding breakthroughs to identify what approach shows the best performance. The focus on the conduciveness for the industry seems to be low. More on this in the *Questionable Aspects*.

## When should you apply Fault Prediction?

The main benefits of Fault Prediction are [2, p. 4626]:

- Reaching a highly dependable system.
- Improving test process by focusing on fault-prone modules.
- Selection of best design from design alternatives using object-oriented metrics.
- Identifying refactoring candidates that are predicted as fault-prone.
- Improving quality by improving test process.

The literature does not state when you should apply this process, but abstracting from the benefits one can say that the intended result can be a high critical system with a low level of fault-tolerance.

This leaves me with the unanswered question when a system is critical enough to require applying fault prediction. The literature mentions companies such as NASA are mentioned that apply this tooling [2, p. 4626] [3, p. 7347]. The importance to mitigate risks for a company like NASA is much higher than the software companies that build simple web services. I can only assume that it is less likely to apply this process in environments where faults are 'allowed'. For example, software to scrape a website has a much higher fault-tolerance, since it is easy to re-execute these kinds of software.

Another consideration that you should keep in mind is when using fault prediction is the possible change in resource allocation. As mentioned in the first section, the process will give you insight on parts of the software that may require more effort in maintenance. The researched literature only states types of fault prediction and their

reliability. No author elaborates on the actions, which a team should apply when having the information where defect may occur. Since it is prediction, the team only knows that defects may occur. Should the team focus on this prediction, or should the team focus on their current backlog, which probably includes concrete defects? Of course, when there are no bugs and the team runs in a NASA-like environment, then the focus of the team may shift or expand. But should the team allocate new team members on mitigating the risks in these 'faulty' modules?

To decide when to use fault prediction a team or company must decide if the risk mitigation is worth it. When a risk turns in to a problem it may have a large impact. For example, an exploding space shuttle. Not only the costs, but also people who might get hurt should be included in this consideration.

## Questionable Aspects

The literature left me with some questionable aspects.

### Severity over Quantity

The literature focuses on the quantity of bugs and where they are likely to occur. The severity of a single defect might be much greater than a group of other defects. Think about a 500 response of a web server instead of the required 404 due to some exception in comparison to a security issue where a user might drop the whole database. The first may occur in greater numbers in an API module, where the second might occur only once in a database module. The fault prediction would push the team's focus towards the API module based on the quantity of defects, which might be less important. In the encountered research, defect severity was not taken into account, which in my opinion is an important aspect in prioritisation.

### Minimum Project Size

A lot of big data sets are used in the research of the different papers. With the researched papers I still have no idea on what environment it is applicable to do fault detection. What would the minimum lines of code be of the project, does it apply on greenfield projects and/or are their even restrictions on the code base?

### Combining Approaches

Both in Catal et al. [3] and Nachiappan et al. [4] the covered research focuses on a certain type of metrics or a fault detection level. What remains an uncertainty is the effect of combining different metric types or fault detection levels. For example, I am curious if it is possible to reach a higher efficiency rate when the 'Organizational Structure' approach is combined with 'Code Churn' or 'Code Complexity'. I question if the industry could benefit more from this research when it was more focussed on combining approaches to improve fault detection efficiency, instead of the decomposition of fault detection types that is currently performed as described in the literature.

### Post Risk Reduction

Suppose a team has used fault detection by other metrics than code coverage. They reduce the risks of the components that are likely to contain defect by adding a bunch of tests. What will happen if the team applies the fault detection again? Does the fault

detection indicate other parts of the software that are likely to fail? I think it will not. The fault detection will start addressing other components of the software when the metrics change. I wonder how this will change, but I guess this depends on the metric suite.

### Conclusion

Fault prediction can identify parts of software that may contain a high amount of defects. Teams that build low fault-tolerance software can benefit from this information by allocating more/other resources. With different metrics, on different levels it is possible to show these spots with a reasonable amount of certainty. Machine learning is a more popular method than statistical methods to detect faults [3, p. 7347] and Organizational Structure seems to be a good identifier to use as metric [4]. I would only apply fault detection when I would be working on a high-risk software project. Personally I think that on other software projects it only adds more workload on the existing backlog.

## References


[1] Martın Abadi et al. "TensorFlow: Large-scale machine learning on heterogeneous systems, 2015". In: *Software available from tensorflow.org* (2015).

[2] Cagatay Catal. "Software fault prediction: A literature review and current trends". In: *Expert Systems with Applications* 38.4 (2011), pp. 4626 –4636. ISSN: 0957-4174. DOI: http://dx.doi.org/10.1016/j.eswa.2010.10.024. URL: http://www.sciencedirect.com/science/article/pii/S0957417410011681.

[3] Cagatay Catal and Banu Diri. "A systematic review of software fault prediction studies". In: *Expert Systems with Applications* 36.4 (2009), pp. 7346 –7354. ISSN: 0957-4174. DOI: http://dx.doi.org/10.1016/j.eswa.2008.10.027. URL: http://www.sciencedirect.com/science/article/pii/S0957417408007215.

[4] Nachiappan Nagappan and Thomas Ball. "Evidence-Based Failure Prediction". In: *Making Software: What Really Works, and Why We Believe It*. 1st ed. O'Reilly, 2011. Chap. 23, pp. 415–434.