DevOps and Machine learning:
Can the efficiency of the DevOps workflow in a company be improved by machine learning?

Marcus Jonsson Ewerbring

May 28, 2020

Contents

1 Introduction

DevOps is getting more popular by companies, where the goal is to minimize the development time and provide well-functioning systems [10]. Machine learning (ML) is a collection name for different learning methods that can predict output patterns from input patterns [20]. This ability makes it possible for the methods to find patterns in data that is hard to detect, which can lead to efficient solutions to programming problems.

Currently, there is no standardized DevOps model, however, it is commonly illustrated by the infinity symbol. The infinity symbol is divided in to 7 different steps, plan, create/building, verify/testing, package, release, configure and monitor [11]. These steps form a ***workflow*** together where each part is essential in order to maintain an efficient workflow, see figure 1.

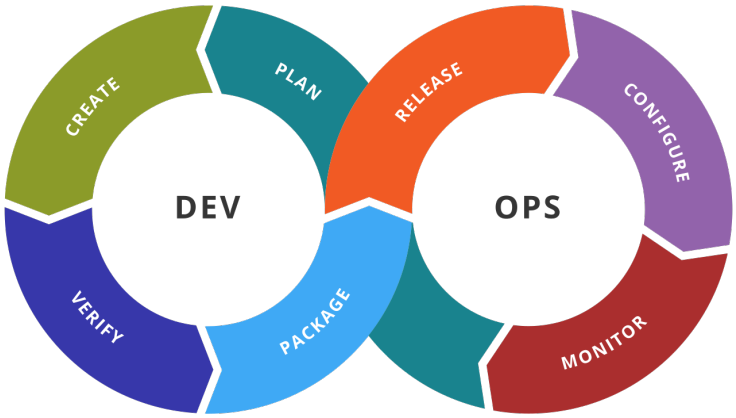

Figure 1: Illustration of the different parts in DevOps, the image is taken from Wikipedia Commons [8].

With an efficient workflow a company can rotate through the different steps in the DevOps model faster, which enables them to deploy products and updates to the software more often.

In order for a company to have a efficient workflow it is important that all steps in the DevOps model works efficiently. One example on a product that attempts to increase the efficiency of the step *planning* is Azure DevOps Labs[17]. Azure DevOps Labs has a tool that enables the companies to set up a online delivery plans for a development team. Another example is the tool Randoop [19] that can generate unit tests to code in order to increase the likelihood of detecting bugs in the code. Randoop would then increase the efficiency of the step *verify*.

Why machine learning
If there already exists tools that can improve the workflow of DevOps in a company, what benefits could machine learning give? I think that the main contribution of machine learning methods is the ability to see patterns in data

and create solution that otherwise would be to hard to code. One example is natural language processing (NLP) which is a machine learning method used for interpretation of the human language [13]. This method could be used to help programmers to spell and write more understandable code or translation of documents.

What is the aim of this essay?
This essay aims to present existing solutions that can improve the efficiency of the DevOps workflow in a company and my own ideas on how this can be achieved by machine learning. The applicability of integrating the suggestions into a company are evaluated and discussed. Because of DevOps being a broad area, many parts of it could potentially be improved by machine learning, I limit this essay to the following subcategories.

- Software monitoring.
- Anomaly detection.
- Software testing.

Software *monitoring* is a step in the DevOps workflow where the software is being monitored in order to check it's performance. This can be done in various ways like monitoring the log outputs of a software or keeping track of the end users experience of the product.

Anomaly detection is a system that can find unusual patterns in data. Anomaly detection are often used in intrusion detection systems (IDS) which monitors a network in order to detect attacks or other malicious activity [18]. These systems could avoid downtime in the DevOps workflow by preventing hackers from disrupting the system.

The category software testing is about testing a program in different ways in order to identify bugs and errors in the code [3]. Machine learning can benefit this category by presenting alternate ways to generate tests cases for code, which could potentially increase the workflow in the *verify* step.

2 Suggested model

In this section I suggest a model on how the DevOps workflow could be improved by machine learning techniques. I suggest that a system consisting of several machine learning assisted tools should be investigated. In figure 2 my suggestion is illustrated and which part of the DevOps workflow it could improve.

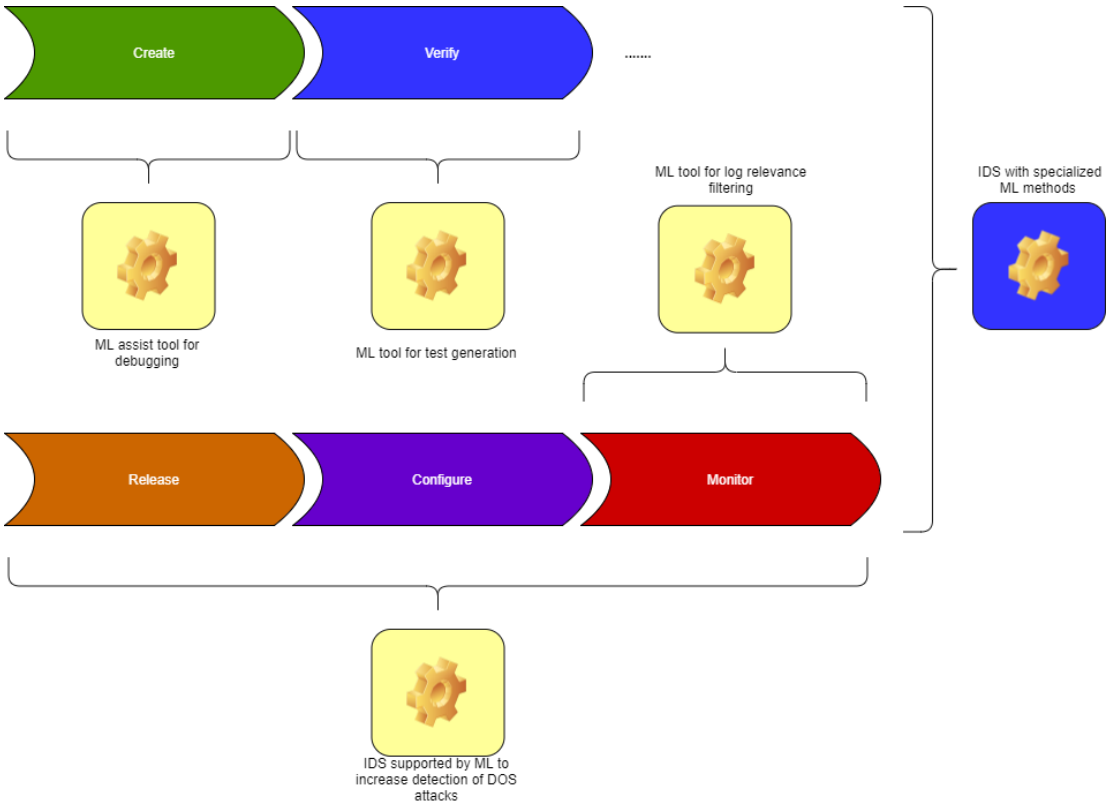

Figure 2: Illustration of suggested improvements

Figure 2 is illustrating 5 tools that could improve different parts of the DevOps model. The yellow boxes represents solutions that I found supporting research papers on how it could be solved. Blue boxes represents methods that I think could be used to improve the DevOps workflow. The improvements to each section are discussed in the following sections and how it could be implemented. Existing implementation to solve these problems and what the difference is compared to the machine learning method is presented.

ML assist tool for debugging
A tool based on machine learning for debugging could be used to help the programmer identify bugs and malfunctioning code early. Detecting and fixing bugs is time consuming and costs money [1]. If this process could be automated or partially automated, it could increase the efficiency of the DevOps step *create*

and then the general DevOps workflow in the company.

In a report from Lionel C. Briand [4] the authors present a machine learning algorithm called RUBAR that can be used for debugging/fault localization. RUBAR is a rule-based statement ranking version of a machine learning algorithm called decision tree [5]. The machine learning algorithm scans code and learns conditions that results in application failure and assumes that test cases failing under similar conditions all have the same error. However, the author points out that the method needs some guidance in form of categories and choices that can result in failure. With given rules the machine learning method could detect error with an accuracy of more than 90%. [4]

How is this problem solved by tools not using machine learning? An existing tool called IBM Rational Software Analyzer[14] can analyze the code and find bugs caused by programming patterns that the company want to avoid. These patterns have to be specified by rules that are predefined by the program but can be supplemented by custom rules from the user. However, the difference between this tool and the proposed one is that the machine learning tool tries to learn unwanted behavior and construct output of that. While the tool from IBM uses static rules to find unwanted behavior.

ML tool for test generation
Testing of software is essential in order to reduce the amounts of bugs in the code and construction of quality code. Writing tests could be considered difficult and time consuming. Therefore, I suggest that the system have support for test generation that is supported or based on machine learning.

Lionel C. Briand [4] proposed a way to automate black box testing[15] with MELBA which means machine learning based refinement of black box. Black box testing works by sending input parameters to a program and look if the output corresponds to the expected output. In black box testing the tester does not know the internal structure of the program. The MELBA method requires an input consisting of a test suite and category-partitioning. Where category-partitioning are different properties of parameters that can influence the tests. The MELBA method could then connect test problems like misclassification, unused categories, etc to different categories that could be a cause to the problem.

There exists applications like Randoop and Evosuite that can automatically generate test cases. Evosuite uses mutation testing in order to achieve high code coverage, for code written in Java [12]. The approach that Briand suggest is not limited to a specific coding language. I consider it to be a tool that could increase performance in existing test generation programs. It could also be used as a tool to construct a personalized test generator for a company's product.

IDS supported by ML to increase detection of DOS attacks
The model that I suggest contains an IDS supported by machine learning in order to protect the system against external attacks. In a report from Amira Sayed A. Aziz, et al [2] the authors attempted to build a intrusion detection system (IDS) with high accuracy at successfully identify an attack. The system

that Aziz, et al built consists of 3 parts specified below.

- Layer 1, feature selection.
- Layer 2, detector generation.
- Layer 3, classification of anomalies by different ML methods.

In layer 1 the authors used principal component analysis (PCA) [22], in order to determine which unique features the training data contains. Layer 2 generates detectors that discriminate between normal behavior and anomalous behavior, this was done by a genetic algorithm [21]. In layer 3 the behavior classified as anomaly is inputted to ML method to confirm if it is an attack or not.

The implementation could detect a denial of service (DOS), probe attack, R2L attack and U2R attack. The authors noted that different implementations of decision tree gave in general the best classification accuracy, however multi-layer neural network [9] gave best result for R2L attacks. The system could detect a DOS attack with 81.97% accuracy, prob attack 65,42% accuracy, R2L attack 33.33% accuracy and U2R attack 20.35% accuracy.

The IDS system by Aziz, et al could detect DOS attacks with high accuracy. By combining the solution by Aziz, et al with an existing IDS it could increase the probability of detecting a DOS attack

IDS with specialized ML methods
Since the machine learning method from Aziz, et al that were presented in the previous section were able to detect one type of attack well, I suggest that their method should be used in a learning technique based on ensemble learning [6]. Ensemble learning is when several machine learning methods or so-called *weak learners* are combined in to one. All the separate methods can be specialized in detecting one type of intrusion. These ML method should be used as a support for a IDS system to increase its detection of threats. If the IDS is capable to detect several kinds of intrusion it could give more protection to the entire company which could benefit the entire DevOps workflow by reducing downtime.

ML tool for log relevance filtering
When a company deploy a software and monitors it's performance, they usually get a lot of log-reports from the program. It can be hard for the programmers and maintainers of the software to find what data that is important and not. This could prolong the process to find a root cause to a problem in the software. Because of that I suggest that the model has a tool for filtering relevance in the log data.

In a report from Zawawy et al [23] the authors try to solve this problem with root cause analysis (RCA). They constructed a framework which has log data, qualifiers and a goal model. The goal model represents the requirements of the monitored software, where the authors point out that this can be obtained by reverse engineering the monitored software. The program outputs a stream of

logical literals that represents the process of events. This can give the users a combined indication on what combination of parameters i causing the crash.

The problem with the solution from Zawawy et al is that their model is dependent on the amount of variables in the log-data and a constructed goal model. If the log-data in an implemented system is inadequate, the model can return an empty string as output, which was mentioned by the authors as a problem. Therefore, I suggest a deep learning model as a supplement to Zawawy et al solution. The deep learning model can monitor the different logs in a system and use a self-orginizing map (SOM) [16] to find the variables in the log-data that seems to affect the systems performance. SOM has the ability to minimize the amount of features in the input data needed to classify the data. A multi-layer perceptron network (MLP) [9] could then be used to map the found features to the constructed goal model, see figure 3.

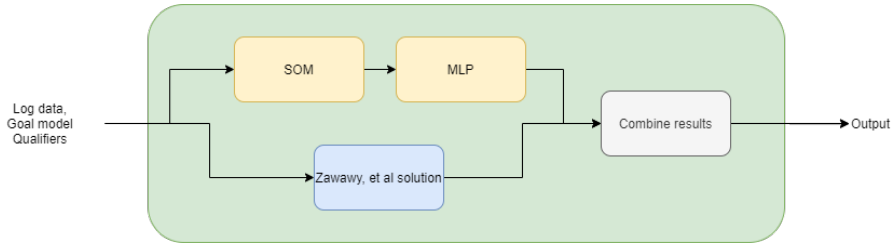

Figure 3: Combination of Zawawy et al solution an my suggestion

The output from Zawawy et al algorith could be compared with the output from the MLP. If the MLPs output a value with a corresponding low probability it would mean that the network is unsure about it's answer and Zawawy et al solution should be prioritized. However, if the output from Zawawy et al is an empty string, the solution from the MLP could be prioritized. I think that this solution could solve the problem of to few or spread variables in the logs that Zawawy et al presented.

3 Applicability

In section 2 a model that could increase performance of the DevOps workflow were presented. In this section it is discussed if this model is applicable in the industry or at a company and what the benefits of using it is. It is motivated why we should implement the model and what the pros and cons are.

3.1 Applicability of the model

In this section the applicability of the different parts of the model presented in 2 is discussed.

ML assist tool for debugging & ML tool for test generation
A model on how the efficiency of debugging and automation of test generation could be increased by machine learning were presented by Lionel C. Briand. He suggested a approach that identifies causes of failed tests and a way for the debugging tool to find suspicious statements in code that could result in a failure of the program. However, the author mentions that both requires some kind of guidance in order to work properly. This affects the applicability of the program negatively because the company needs to identify these categories and create the machine learning method. If they successfully manage to create these categories and machine learning method, the company could save time by having the debugging and testing time reduced.

IDS supported by ML to increase detection of DOS attacks
The technology proposed by Aziz, et al in section 2 could detect a DOS attack with 81.97% accuracy, this could improve the release, configure and monitor step of the DevOps model by preventing system downtime. The technology was not equally successful in discovering other types of attacks. This technology could be used as an addition to existing IDS in order to increase detection rate of DOS attacks. It could result in a patch for existing IDS which would make it applicable for all the users of that IDS. If more research is conducted in the field, the detection rate for other types of intrusion may be increased and improve DevOps.

IDS with specialized ML methods
This suggestion is a modified version of *IDS supported by ML to increase detection of DOS attacks* were I suggested that Amira Sayed A. Aziz, et al, method should be used in a ensemble network and specified on different kinds of attack. This solution need to be explored, examined and tested before it could be applied in to a company. However, if it is successful it could as the previous solution be delivered with a path to an existing IDS.

ML tool for log relevance filtering
The log relevance filtering method presented in 2 are a combination of Zawawy, et al, RCA based filtering and my suggestion on using SOM and MLP. The idea of my suggestion was to make the method more adaptable to already existing

software with varying formats on the logs. However, as mentioned by Zawawy, et al, the solution needs a goal state as input. If a company should adopt this solution, they need to construct a goal model for each system they want to apply it on. This process could potentially be atomized, however, this has not been evaluated in this essay. The benefit of this model is that it should be applicable to many systems that uses logs to monitor performance. However, the logs must contain some relevant data in order for this method to be useful. The machine learning part of the solution is a suggestion from me and need to be thoroughly examined before applying it to a company, however, it could result in a solution that minimize the amount of log data the programmers need to read in order to find a cause to an error.

Summary of applicability
The common problem of the proposed model is that the suggested machine learning methods need to be constructed and tested before they can be applied to a company. It would require time and resources to evaluate these methods. Dependencies of the method that was not detected during the design of the model could be found. However, the model contains several parts and they work independently, if some parts of the model would not be applicable to a company it could be removed.

3.2 Why should we implement the model

Some part of the suggested method is meant as a supplement to already existing tools like test generation tools as Randoop and Evosuit. If there already exist working tools why should we spend a lot of money to implement a new?

Tools like Randoop and Evosuit are adapted to one language with the aim to increase it's code coverage. There exist similar tools for other languages, for example test-generator 0.1.2 for python. The suggestion in this essay is not dependent on the language of the code, which makes it more adaptable to the situation. But what if the software is written in Java, should we ignore this solution and use Evosuit instead? If several tools is available to generate test it could increase the chances of finding bugs. The suggested tool in this essay could find a case that Evosuit or Randoop did not detect, which could increase the code coverage of the tests.

This suggestion also mentions a potential improvement to IDS systems and protection for the system. These machine learning methods are automated and could potentially increase the detection of threats to a system. If a threat remains undetected it can harm a system and cause downtime which could result in loss of money.

This model could increase the efficiency of the DevOps workflow in many of the steps. In a paper from Constantine Aaron Cois, et al, [7] the authors concluded that heavy automation of can improve the effectiveness of the DevOps model. By automating methods that are time consuming and distracting for the development team a more efficient workflow could be achieved.

3.3 Pros and cons

In table 1 the pros and cons of the suggested model are presented.

Pros	Cons
• The model could benefit several steps of the DevOps model. • It could be combined with already existing solutions in order to increase performance. • It could save money by reducing downtime and damage done by different attacks. • The area is new, improvements in the area i likely to appear. • This model could improve existing tools.	• It requires many different machine learning methods to be implemented. • The detection rate for DOS attacks were 80%, this results in that 20% of the attacks is undetected. • The area is new which could result in that this model becomes outdated. • It already exists many good tools for test generation, debugging and IDS.

Table 1: Pros and cons of the suggestion

4 Conclusion

In this essay it was discussed if the DevOps workflow could be improved by machine learning. The essay was limited to three subcategories regarding DevOps: software monitoring, anomaly detection and software testing. A model based on machine learning was suggested. The model consists of several machine learning methods supporting debugging, automatic test-generation, potential improvements to IDS and software log-filtering. The different parts of the model were meant as a support of existing solutions or other proposed solutions.

The applicability of each method on a company was investigated. The solution for debugging and test generation could be used, however, the methods needs guidance categories to work properly which had to be created by the company. The IDS with ML support system could be used, however only for detection of DOS attacks. It was suggested to combine this with existing IDS to construct a patch to increase applicability. A suggestion was proposed where several versions of the previous techniques were combined to decrease downtime and increase security of a company. A method for log-filtering was proposed

and combined with the machine learning methods SOM and MLP in order to make the solution more adaptable to systems with few or spread variables in the logs. The log-filtering needs to be tested thoroughly before usage in a company, but if it is approved it could increase the efficiency of the monitor step in the DevOps model.

It was concluded that machine learning and the suggested model could have the capability to increase the DevOps workflow in a company. However, since more research is needed in the field, the model could be hard to implement.

References

[1] Andrea Arcuri. "On the Automation of Fixing Software Bugs". In: *Companion of the 30th International Conference on Software Engineering*. ICSE Companion '08. Leipzig, Germany: Association for Computing Machinery, 2008, pp. 1003–1006. ISBN: 9781605580791. DOI: 10.1145/1370175.1370223. URL: https://doi.org/10.1145/1370175.1370223.

[2] A. S. A. Aziz et al. "Multi-layer hybrid machine learning techniques for anomalies detection and classification approach". In: *13th International Conference on Hybrid Intelligent Systems (HIS 2013)*. 2013, pp. 215–220.

[3] A Bertolino. "Software Testing Research: Achievements, Challenges, Dreams". eng. In: *Future of Software Engineering (FOSE '07)*. IEEE, 2007, pp. 85–103. ISBN: 0769528295.

[4] L. C. Briand. "Novel Applications of Machine Learning in Software Testing". In: *2008 The Eighth International Conference on Quality Software*. 2008, pp. 3–10.

[5] L. C. Briand, Y. Labiche, and X. Liu. "Using Machine Learning to Support Debugging with Tarantula". In: *The 18th IEEE International Symposium on Software Reliability (ISSRE '07)*. 2007, pp. 137–146.

[6] Gavin Brown. "Ensemble Learning". In: *Encyclopedia of Machine Learning* (2010). URL: http://www.cs.man.ac.uk/~gbrown/research/brown10ensemblelearning.pdf.

[7] C. A. Cois, J. Yankel, and A. Connell. "Modern DevOps: Optimizing software development through effective system interactions". In: *2014 IEEE International Professional Communication Conference (IPCC)*. 2014, pp. 1–7.

[8] Wikipedia commons. *File:Devops-toolchain.svg*. June 2019. URL: https://commons.wikimedia.org/wiki/File:Devops-toolchain.svg.

[9] Pratap Dangeti. *Statistics for Machine Learning*. Packt Publishing Ltd, July 2017. ISBN: 978-1-78829-575-8.

[10] Christof Ebert et al. "DevOps". In: *IEEE Software* 33.3 (2016), pp. 94–100. ISSN: 0740-7459.

[11] IBM Cloud Education. *DevOps*. URL: https://www.ibm.com/cloud/learn/devops-a-complete-guide.

[12] Gordon Fraser and Andrea Arcuri. "EvoSuite: Automatic Test Suite Generation for Object-Oriented Software". In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ESEC/FSE '11. Szeged, Hungary: Association for Computing Machinery, 2011, pp. 416–419. ISBN: 9781450304436. DOI: 10.1145/2025113.2025179. URL: https://doi.org/10.1145/2025113.2025179.

[13] Julia Hirschberg and Christopher D Manning. "Advances in natural language processing." eng. In: *Science (New York, N.Y.)* 349.6245 (2015), pp. 261–266. ISSN: 00368075. URL: http://search.proquest.com/docview/1697220733/.

[14] IBM. *Static analysis IBM Rational Software Analyzer, Getting started*. May 2019. URL: https://www.ibm.com/developerworks/rational/library/08/0429_gutz1/.

[15] Mohd. Ehmer Khan and Farmeena Khan. "A Comparative Study of White Box, Black Box and Grey Box Testing Techniques". In: *International Journal of Advanced Computer Science and Applications*. Vol. 3. 6. 2012, pp. 12–15.

[16] Teuvo Kohonen. "Essentials of the self-organizing map". In: *Neural Networks* 37 (2013). Twenty-fifth Anniversay Commemorative Issue, pp. 52–65. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2012.09.018. URL: http://www.sciencedirect.com/science/article/pii/S0893608012002596.

[17] Azure DevOps Labs. *Managing project schedules across teams with Delivery Plans*. URL: https://azuredevopslabs.com/labs/azuredevops/deliveryplans/.

[18] John Mchugh, Alan Christie, and Julia Allen. "Defending Yourself: The Role of Intrusion Detection Systems". eng. In: *IEEE Software* 17.5 (2000), pp. 42–51. ISSN: 0740-7459.

[19] Carlos Pacheco and Michael D. Ernst. "Randoop: Feedback-Directed Random Testing for Java". In: *Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion*. OOPSLA '07. Montreal, Quebec, Canada: Association for Computing Machinery, 2007, pp. 815–816. ISBN: 9781595938657. DOI: 10.1145/1297846.1297902. URL: https://doi.org/10.1145/1297846.1297902.

[20] Gareth James Daniela Witten Trevor Hastie Robert Tibshirani. *An Introduction to Statistical Learning*. 2013. ISBN: 9781461471370.

[21] Darrell Whitley. "A genetic algorithm tutorial". eng. In: *Statistics and Computing* 4.2 (1994), pp. 65–85. ISSN: 0960-3174.

[22] Svante Wold, Kim Esbensen, and Paul Geladi. "Principal component analysis". In: *Chemometrics and Intelligent Laboratory Systems* 2.1 (1987). Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists, pp. 37–52. ISSN: 0169-7439. DOI: https://doi.org/10.1016/0169-7439(87)80084-9. URL: http://www.sciencedirect.com/science/article/pii/0169743987800849.

[23] Hamzeh Zawawy, Kostas Kontogiannis, and John Mylopoulos. "Log filtering and interpretation for root cause analysis". eng. In: *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–5. ISBN: 9781424486304.