Research Article

**Change Impact Analysis Case Study for Aviation: Mutation Testing**

Nurbanu Hınık1*,2, Özcan Çağırıcı1, Ufuk Sakarya3

1 Turkish Aerospace Industries, İstanbul, Turkey, ORCID: 0000-0001-5658-7291, nurbanuhinin1@gmail.com.tr, ORCID: 0000-0003-2215-3684, ozcan.cagirici@gmail.com
2 YILDIZ Technical University, Graduate School of Science and Engineering, Department of Avionics Engineering, İstanbul, Turkey
3 YILDIZ Technical University, Faculty of Applied Sciences, Department of Aviation Electrics and Electronics, İstanbul, Turkey, ORCID: 0000-0002-8365-3415, usakarya@yildiz.edu.tr
* Correspondence: nurbanuhinin1@gmail.com

(First received March 23, 2022 and in final form June 01, 2022)


**Abstract**

As the complexity of modern software systems increases, changes in software have become crucial to the software lifecycle. For this reason, it is an important issue for software developers to analyze the changes that occur in the software and to prevent the changes from causing errors in the software. In this paper, mutation testing as software test analysis is examined. Mutation tests have been implemented on open-source Java projects. In addition, for aviation projects, emphasis is placed on performing change impact analysis processes in compliance with the certification based on DO-178C processes.

**Keywords:** Change Impact Analysis, Mutation Testing, Software Testing

1. **Introduction**

Change is inevitable throughout the software development lifecycle. Software configuration management is the process that exists to systematically manage, control, and regulate changes in documents, code, and other assets throughout its entire lifecycle. When considering the aviation industry, the software configuration management process is critical for both military and civil aviation applications as the complexity of avionics systems increases. The main purpose of the software configuration management process is to increase production while reducing errors. One of the key activities of software configuration management is change control. Along with software change control, all factors associated with a proposed change in the software are evaluated, change priorities are determined and incorrect/undesirable changes are prevented. A change in software,
if left unchecked, can affect all assets, and have catastrophic consequences. Determining
the software impact of changes in avionics software development projects is also in the
field of system engineering. In this context, the concept of Trade Studies (trade-off
analysis) is defined as a decision-making approach used by integrated teams to make
alternatives and resolve conflicts within the framework of systems engineering [8].

RTCA DO-178C, which is a globally accepted guidance document in avionic
software development projects, defines objectives for each process of the software
lifecycle. In this direction, the impact of changes in the software on the whole software is
expressed as "change impact analysis" in the DO-178C guidance document. According to
the DO-178C guidance document, changes in an application or development
environment should be identified, analyzed, and re-validated [5]. It is recommended to
counter a change impact analysis of the changes that occur in the software, with a method
determined in the Change Review phase [5]. Basically, change impact analysis is required
for every change [15]. Although the extent to which change impact analysis should be
applied for each change that occurs is described in the DO-178C guidance document, it
is not specified which methods will be used.

There are other approaches on this subject. It has been shown that graph
measurements can be used to predict error-prone versions of software, with a graph
method that examines the structure and evolution of software [3]. The release
management process, which is an important part of the software change control process,
has been modeled as seven stages, and the activities to be implemented in these stages
and the roles of individuals have been defined [7]. In the study where a generative model
of software dependency graphs was designed (GD-GNC) and the structure of artificial
graphs created with this model were compared with real software graphs, it was
examined that there is a common arrangement in software dependency graphs [12]. In
the study conducted to characterize the graph-based model to be used in software and
systems engineering, evaluations were made on many graph metrics and sample models
[18]. Automatic generation of graph models has become an important approach in
various applications of software and systems engineering. In this context, a technique is
proposed to derive an automatic graph model [17]. An interactive visualization tool
based on call graphs has been developed to help software developers understand the
software system by focusing on specific parts of software systems separately [1]. The
impact of software language selection on software quality in software development has
been investigated using open-source software projects [4]. An open-source tool called
GraphEvo, which can automatically generate and visualize color-coded call graphs, has
been developed to help software developers better understand the software and monitor
the changes that occur in the software [19]. A small change in the software system can
produce large local and non-local ripple effects throughout the software system. For this
reason, a large-scale mature software system was examined and the change in entropy
value was observed as the software evolved [9]. A protocol based on mutation testing is presented to quantitatively evaluate the impact analysis of errors that occur as the software evolves [13]. Change impact analysis has two major challenges, scaling to the size of software systems consisting of thousands of modules, and accuracy of impact estimates. The accuracy of the impact estimation has two points: whether the predicted impacted items are actually impacted and whether all of the actually impacted items are predicted. Accordingly, learning algorithms have been studied to improve impact estimation based on call graphs [11]. Finally, open-source software projects and determined mutation operators have been studied on the estimation of effect propagation. In this study, impact estimation was analyzed based on call graphs [14]. As seen in all these studies, every change in the software should be analyzed and controlled.

In this paper, in order to emphasize the importance of change impact analysis in software systems, mutation testing was applied with the mutation operators we determined for open-source software projects. The extent to which software systems are impacted by the changes we have made has been evaluated. We would like to emphasize the importance of change impact analysis for an Avionics software development project compatible with DO-178C processes. The projects we analyzed in our study are open source and not aviation projects. What we want to show in this study is that change effect analysis can be applied using mutation testing.

The rest of the paper is planned as follows. The related works are presented in Section 2. The proposed study is introduced in Section 3. The experimental results are demonstrated in Section 4, and finally, the paper is concluded in Section 5.

2. Related Works

2.1. DO-178C

It is a guidance document prepared to be used in the software development process in the systems and equipment of the “flying” vehicles. The purpose of the DO-178C guidance document is to ensure that the software in a system or device is compatible with airworthiness requirements at the desired level. It defines the objectives of the processes used throughout the software lifecycle, the activities that must be done to achieve these objectives, and the outputs of these activities. Software lifecycle processes defined by DO-178C guidance document are defined as software planning, software development and software lifecycle support processes; software verification, software quality assurance, software configuration management, certification liaison processes. Every process is related to each other [5].
The software configuration management process includes activities such as defining software configuration items, providing their controls, keeping configuration management records, and managing the software change process. This paper focuses on the software change management process, which is an important activity of the DO-178C guidance document’s software configuration management process.

Change can occur at any process of the software development process and is a core feature of software. Changes can occur for many reasons, such as adding or changing functions, fixing a bug, performance improvements, improving processes, changing the environment. Without effective change management, chaos ensues. One of the tasks of an effective change management process is to perform change impact analysis. To properly plan the resources required to implement a change and the amount of revalidation required, the impact of the change must be analyzed [15].

### 2.2. Change Impact Analysis

For safety-critical systems, software changes are a way of life. In the process of designing software products, care should be taken to ensure that the software is maintainable and upgradeable. When software changes occur in safety-critical software systems, they must be carefully planned, analyzed, implemented, controlled, and verified. In a certified product, a change impact analysis is required for approved and
modified software [15]. The analyzes that should be included in a change impact analysis are described below [15][16]:

Traceability analysis: Identifies software items that may be directly or indirectly affected by the change. Traceability analysis aims at determination of the impact of the change on the software project. It is important to be able to identify both replaced and impacted software items.

Analysis of memory margins: It is carried out to ensure that the memory allocation requirements are met and protected. It cannot be completed until the change is applied. It can be predicted early, but the actual impact assessment is completed after implementation of the change.

Analysis of timing margins: It is carried out to ensure that timing requirements are met and protected. It cannot be completed until the change is applied.

Analysis of data flow: Identifies the negative effects caused by changes in the data flow.

Analysis of control flow: Identifies adverse effects caused by changes in control flow.

Input/output analysis: Identifies the impact of software changes on the input and output requirements of the product.

Development environment and process analysis: Identifies any changes that may affect the software product.

Operational characteristics analysis: It ensures that changes that may affect system operation do not adversely affect the system.

Certification maintenance requirements analysis: In case of software change, it identifies whether a new or changed certificate needs maintenance. If each change made in the software affects a certificate maintenance requirement, it is included in the scope of change impact analysis.

Partitioning analysis: Guarantees that changes to the software do not affect the mechanisms included in the design.

The software lifecycle data impacted by the change should be documented as part of the change impact analysis. It is critical for software projects to have a well-defined change impact analysis process [5]. There are many change impact analysis methods that have been used for years in the software configuration management process.

### 2.3 Mutation Analysis

Mutation testing is a method for increasing the code coverage of test suites. In this approach, it is built on making numerous copies of a program, known as mutants, in which one (or more than one) minor changes are added. By running test suites on each mutation, it is possible to identify source code portions that require more testing. It is modified a portion of the code (i.e., mutate the code) in each time, a little flaw is added
that test suites should identify. If the change is not identified, it implies that test suites are not adequately covering the change [10]. If all of the test pass, the mutant is considered to be alive, which suggests that the test coverage should be improved. In the contrary instance, if at least a single test fails, the mutant is called by killed, implying that the test covers the mutated region [10].

A mutation operator specifies the type and method in which items are changed. Mutation testing is an example of what is commonly referred to as error-based testing. In other words, it entails the generation of test data with the goal of detecting particular errors or classes of errors. A program undergoes a huge number of simple modifications (mutations) one at a time. Then, test data that identifies the mutated versions from the original version must be found [20].

3. Proposed Case Study: Mutation Testing

The change impact analysis process, which is an activity of the software change management process, which plays a critical role in the software development life cycle, has been examined. One of the most successful ways to protect software systems from erroneous changes is to improve the quality of software test cases. Based on this, it is proposed to apply mutation testing in order to prevent erroneous changes in software systems. Traditional test coverage only measures what code is run by software tests. It does not check if the tests can actually detect errors in the code that is being executed. For this reason, it is proposed that mutation test coverage percentages measured by applying mutation testing to a software with determined mutation operators can be used as an effective method to measure the durability of projects. In Section 4, mutation test coverage results are shown by applying mutation testing to three identified open-source Java projects.

4. Experimental Results

4.1. Dataset

To analyze changes that occur in software systems, it is considered that a dataset composed of three Java software packages: Google Gson, Apache Commons Codec, Apache Commons Lang. Table 1 contains the name, the version, the git commit-id, and the number of lines of code (computed using Jacoco [2]) of the software being analyzed.
4.2. Implementation Details

The experiments done in this paper are implemented in the Pitest tool. An open-source tool, Pitest, was used to perform mutation testing [6]. The open source Jacoco tool was used to calculate the line of code value and test coverage percentage of the projects we analyzed [2].

Mutation operators are required for our method. The mutation operator changes only one atomic element. A mutation can include any software source code elements. The mutation operators we used in this paper are listed in Table 2.

Table 1 Statistics About the Project Considered in This Paper

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Version</th>
<th>Commit</th>
<th>Line of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Gson</td>
<td>2.3.2</td>
<td>fcfd397d</td>
<td>17745</td>
</tr>
<tr>
<td>Apache Commons Codec</td>
<td>1.11</td>
<td>7decf21b</td>
<td>50648</td>
</tr>
<tr>
<td>Apache Commons Lang</td>
<td>3.11</td>
<td>75a6aac07</td>
<td>74166</td>
</tr>
</tbody>
</table>

Table 2 List of Mutation Operators Considered in This Paper

<table>
<thead>
<tr>
<th>Mutation Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditionals Boundary</td>
<td>The conditionals boundary mutator substitutes the boundary counterpart for the relational operators, =, &gt;, &gt;=.</td>
</tr>
<tr>
<td>Constructor Calls</td>
<td>The constructor calls mutator switches constructor calls with null values.</td>
</tr>
<tr>
<td>Experimental Argument Propagation</td>
<td>The experimental mutator substitutes method calls with one of its matching type parameters.</td>
</tr>
<tr>
<td>False Returns</td>
<td>Changes boxed and primitive boolean return values with false.</td>
</tr>
<tr>
<td>Math</td>
<td>For either integer or floating-point arithmetic, the math mutator substitutes another operation for binary operations.</td>
</tr>
<tr>
<td>Return Values</td>
<td>The new returns mutator set has taken the place of this mutator. The return values mutator modifies method call return values. A different mutation is used depending on the method’s return type.</td>
</tr>
<tr>
<td>True Returns</td>
<td>True is used to replace primitive and boxed boolean return values.</td>
</tr>
</tbody>
</table>
4.3. Results

Within the scope of the projects we analyzed, the mutation test result in line with the mutation operators we determined is given in Table 3. Table 3 contains the project name, the name of the mutation operator used in this paper, number of classes, percentage of unit test line coverage, classes in which mutation operators are available unit test coverage (Unit Test Line Coverage), percentage of mutation test line coverage, mutation test coverage of functions where mutation operators are available (Mutation Test Line Coverage), percentage of mutation test strength, number of mutation test (Test Strength) of the software being analyzed.

The comparison of unit test coverage values and mutation test coverage values on the basis of the examined projects is given in Table 4. Table 4 contains the project name, the number of lines of code (computed using Jacoco [2]) percentage of unit test coverage and percentage of mutation test of the software being analyzed. Although the result obtained from this comparison seems to have a high unit test coverage, the coverage value decreases as a result of the mutation test. This shows that the unit test coverage value can not enough to detect errors in the source code.

5. Conclusion

Any change in the software may affect the entire software system. Software testing is essential to building an effective and fault-tolerant software system. The emphasis in this paper is the use of mutation testing methods to prevent errors that may occur in the software development life cycle. In this paper, the case study we have created on open-source Java projects shows that by measuring the scope of software tests with mutation testing, we show that mistakes in change situations can be prevented. The importance of applying the change impact analysis process in certification-compliant aviation projects is emphasized in the guidance document DO-178C. Although we have shown our results in this paper through open-source Java projects, our main goal is to increase the software quality by using these methods in safety-critical aviation projects. Our future work is to create change impact analysis methods for DO-178C compatible software by using these methods in open-source aviation projects.

Table 3 Mutation Analysis Results

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Mutation Operator</th>
<th>Number of Classes</th>
<th>% Unit Test Line Coverage</th>
<th>% Unit Test Line Coverage</th>
<th>% Mutation Test Line Coverage</th>
<th>% Test Strength</th>
</tr>
</thead>
</table>

Table 4 Unit Test and Mutation Test Coverage Results
<table>
<thead>
<tr>
<th>Google Gson</th>
<th>Conditionals_Boundary</th>
<th>11</th>
<th>92</th>
<th>1751/19</th>
<th>57</th>
<th>37/65</th>
<th>59</th>
<th>37/63</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor_Calls</td>
<td>37</td>
<td>87</td>
<td>3267/37</td>
<td>63</td>
<td>292/46</td>
<td>97</td>
<td>292/30</td>
<td></td>
</tr>
<tr>
<td>Experimental_Argument_Propagation</td>
<td>23</td>
<td>89</td>
<td>2218/24</td>
<td>54</td>
<td>66/123</td>
<td>61</td>
<td>66/109</td>
<td></td>
</tr>
<tr>
<td>False_Returns</td>
<td>17</td>
<td>88</td>
<td>1492/16</td>
<td>74</td>
<td>51/69</td>
<td>93</td>
<td>51/55</td>
<td></td>
</tr>
<tr>
<td>Math</td>
<td>13</td>
<td>91</td>
<td>1781/19</td>
<td>70</td>
<td>164/23</td>
<td>72</td>
<td>164/22</td>
<td></td>
</tr>
<tr>
<td>Return_Vals</td>
<td>41</td>
<td>88</td>
<td>3463/39</td>
<td>76</td>
<td>559/73</td>
<td>92</td>
<td>559/60</td>
<td></td>
</tr>
<tr>
<td>True_Returns</td>
<td>18</td>
<td>86</td>
<td>2000/23</td>
<td>68</td>
<td>71/105</td>
<td>93</td>
<td>71/76</td>
<td></td>
</tr>
<tr>
<td>Apache Commons Codec</td>
<td>Conditionals_Boundary</td>
<td>38</td>
<td>95</td>
<td>3859/40</td>
<td>67</td>
<td>168/25</td>
<td>67</td>
<td>168/25</td>
</tr>
<tr>
<td>Constructor_Calls</td>
<td>50</td>
<td>94</td>
<td>3830/40</td>
<td>81</td>
<td>268/32</td>
<td>94</td>
<td>268/28</td>
<td></td>
</tr>
<tr>
<td>Experimental_Argument_Propagation</td>
<td>47</td>
<td>95</td>
<td>3970/41</td>
<td>82</td>
<td>416/50</td>
<td>88</td>
<td>416/47</td>
<td></td>
</tr>
<tr>
<td>False_Returns</td>
<td>23</td>
<td>96</td>
<td>1983/20</td>
<td>82</td>
<td>45/55</td>
<td>88</td>
<td>45/51</td>
<td></td>
</tr>
<tr>
<td>Math</td>
<td>34</td>
<td>96</td>
<td>3520/36</td>
<td>92</td>
<td>1354/1</td>
<td>95</td>
<td>1354/1</td>
<td></td>
</tr>
<tr>
<td>Return_Vals</td>
<td>56</td>
<td>95</td>
<td>4367/46</td>
<td>82</td>
<td>696/85</td>
<td>92</td>
<td>696/75</td>
<td></td>
</tr>
<tr>
<td>True_Returns</td>
<td>30</td>
<td>95</td>
<td>2627/27</td>
<td>85</td>
<td>89/105</td>
<td>92</td>
<td>89/97</td>
<td></td>
</tr>
<tr>
<td>Apache Commons Lang</td>
<td>Conditionals_Boundary</td>
<td>60</td>
<td>96</td>
<td>12172/1</td>
<td>55</td>
<td>735/13</td>
<td>55</td>
<td>735/13</td>
</tr>
<tr>
<td>Constructor_Calls</td>
<td>103</td>
<td>96</td>
<td>13782/1</td>
<td>82</td>
<td>798/97</td>
<td>96</td>
<td>798/83</td>
<td></td>
</tr>
<tr>
<td>Experimental_Argument_Propagation</td>
<td>122</td>
<td>96</td>
<td>14153/1</td>
<td>77</td>
<td>1196/1</td>
<td>80</td>
<td>1196/1</td>
<td></td>
</tr>
<tr>
<td>False_Returns</td>
<td>91</td>
<td>96</td>
<td>12158/1</td>
<td>90</td>
<td>528/58</td>
<td>97</td>
<td>528/54</td>
<td></td>
</tr>
<tr>
<td>Math</td>
<td>65</td>
<td>97</td>
<td>11465/1</td>
<td>82</td>
<td>1299/1</td>
<td>83</td>
<td>1299/1</td>
<td></td>
</tr>
<tr>
<td>Return_Vals</td>
<td>177</td>
<td>95</td>
<td>15765/1</td>
<td>88</td>
<td>4126/4</td>
<td>95</td>
<td>4126/4</td>
<td></td>
</tr>
</tbody>
</table>
Table 4 Comparison of Project-Based Unit Test Coverage and Mutation Test Coverage

<table>
<thead>
<tr>
<th>Project Name</th>
<th>Line of Code</th>
<th>Unit Test Coverage Percentage</th>
<th>Mutation Test Coverage Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google Gson</td>
<td>17,745</td>
<td>84</td>
<td>69</td>
</tr>
<tr>
<td>Apache Commons Codec</td>
<td>50,648</td>
<td>97</td>
<td>85</td>
</tr>
<tr>
<td>Apache Commons Lang</td>
<td>74,166</td>
<td>95</td>
<td>81</td>
</tr>
</tbody>
</table>

References


