Software Testing Based on Models – Applying Model-Based Testing on the Example of the Software TiGL

Kaan Akbulut[1] and Johannes Simon[2]

[1] Matriculation number: 7328725
[2] Matriculation number: 7364536
University of Cologne, Albertus-Magnus-Platz, 50923 Cologne, Germany
kakbulut@smail.uni-koeln.de, jsimon10@smail.uni-koeln.de

Abstract. Model-based testing (MBT) is a black-box testing technique that aims to automatically generate and execute test cases by utilizing an abstract system model (Marinescu et al., 2015, p. 91). However, despite of the huge potential benefits of MBT (Schieferdecker, 2012, p. 14), there are still cases, such as the example of TiGL, an open-source software that creates three-dimensional airplane geometries (Siggel et al., 2019, p. 367), where MBT has not yet been used. There is also a lack of concrete practical examples in research. Presenting such an example is the aim of this work, for which we create a prototype. The creation process is divided into four parts: the definition of an MBT workflow based on an online scientific database research, the selection of an MBT tool based on a web-based research, the creation of test models as well as the execution of the tests. One of the main results is the modelling prototype that has been created with GraphWalker in order to test the graphical user interface (GUI) of TiGL in terms of correctly opening and displaying aircraft models as well as correctly selecting and rotating components of aircraft models.

Keywords: Model-based testing, Test automation, Software Engineering, Aircraft.

1 Introduction

1.1 Motivation

Testing software systems is one of the most demanding and important tasks in software development (Marinescu et al., 2015, p. 91). Aim of testing is to ensure that there are no bugs in the software system, but when tests are written manually, they can be incomplete and also time-consuming, especially in industry (Marinescu et al., 2015, p. 91). One possible solution can be model-based testing (MBT), which can be used for automated test designs and generation (Schieferdecker, 2012, p. 14). There already exist various MBT tools, each with different characteristics (Marinescu et al., 2015, p. 91), which are developed by organizations worldwide (*Model Based Testing*, 2020). The potential benefits of MBT could be huge (Schieferdecker, 2012, p. 14),

but there are still cases, such as the example of the software TiGL, where MBT has not yet been used.
MBT has also gathered more attention in research over the years (Marinescu et al., 2015, p. 91). There are many papers that are developing MBT approaches and providing general overviews on MBT, but there is a lack of concrete examples on how MBT can actually be utilized on software systems. In a similar context, Alégroth et al. (2022, p. 37) also notice a lack of experience reports about MBT in practice.
Considering the number of available MBT tools, this leads to the following research question: *Which MBT tool is suitable for testing the software TiGL?*
Our work aims to create a prototype of MBT in the context of TiGL by designing initial test models and executing tests with these models. To be able to create the best possible results, we also conduct an interview with one of the TiGL developers to get better knowledge of TiGL, an online scientific database research to structure our workflow and define the MBT tool selection criteria as well as a web-based research to find and select the MBT tool.
In research, this work could possibly highlight how well the current state of research on MBT fares in practical use. This can then be utilized to identify weaknesses or other suggestions for improvement that can further accelerate research on MBT.
This work can potentially contribute to providing an initial approach for implementing MBT in the development of TiGL. Additionally, developers of other software systems could also take this work as an example and create their own models on this basis. In both cases, it could make testing the respective software much faster (Saket, n.d.) and more reliable (Software Testing Magazine, 2022).

1.2 Research Objective

The main goal of our research is to evaluate and prototype how MBT can be used for testing TiGL. Concretely, we want to evaluate which MBT tool is suitable for this purpose. To achieve this main goal, our research is divided into sub-goals that cover various relevant key areas. The first sub-goal is to implement a structured workflow for our case of testing TiGL with MBT. This is considered done after defining clear steps on how to conduct MBT in TiGL and utilize the MBT tool as well as identify what part of TiGL the MBT tool should test. Then, the focus moves on to selecting the MBT tool we utilize for our work. This sub-goal is considered done when an MBT tool has been selected whose characteristics are suitable for testing TiGL. The next sub-goal is to use the selected MBT tool to create models that represent the requirements of TiGL to be tested in the form of test cases. Here, the sub-goal is considered done if at least one model has been created based on a requirement of TiGL that is described as a test case. The final sub-goal is to execute the previously created test models. It is considered done if these models generate tests that can be run and completed with TiGL as well as provide correct results on whether these tests have been successful. Combining the results of these sub-goals then represent our evaluation and prototype on how to use MBT for testing TiGL.

1.3 Structure/Outline

In Chapter 2, we explain background concepts that do not answer the research question, but are needed to understand the context of this work. This background includes a description of the concept of MBT and the system under test (SUT) TiGL.
In Chapter 3, we describe the method we use for this work. It involves the choice and approach of method with a detailed description on how and why the selected method has been used as well as the selection criteria we use for the MBT tool.
In Chapter 4, we present the results of our work. The results are separated into four sub-chapters that align with our sub-goals defined in Chapter 1.2: “MBT Workflow”, “MBT Tool Selection”, “Test Cases and Modelling Prototype” and “Running Tests”.
In Chapter 5, we discuss the results from Chapter 4. There, we discuss the contribution of this work to research and practice, describe the limitations and make propositions for future research.
In Chapter 6, we conclude this work and summarize our results.

2 Background and Related Work

2.1 Model-Based Testing (MBT)

MBT is a black-box testing technique that aims to automatically generate and execute tests by utilizing an abstract system model (Marinescu et al., 2015, p. 91). This system model represents the intended behaviour of the SUT and/or its environment (Utting et al., 2012, p. 297), while this intended behaviour is usually expressed through requirements, specifications and other similar documents (Marinescu et al., 2015, p. 93). Meanwhile, the type and appearance of the system model can differ based on the domain, context or purpose (Alégroth et al., 2022, p. 2). MBT can help improving the overall testing strategy by reducing errors and making it more efficient (Saket, n.d.) as it aims to avoid problems caused by defining tests traditionally, such as the reliance on the level of skills of the software tester (Software Testing Magazine, 2022). However, an important challenge is the difficulty and time required to create the system models (Saket, n.d.). In order to create appropriate system models, it is first necessary to understand the SUT (Saket, n.d.) in terms of what features it has and how it works. In the case of our work, for example, it would very difficult to create appropriate system models without having any knowledge about TiGL.

2.2 TiGL

TiGL is an open-source geometry modeler developed primarily at the German Aerospace Center (DLR), where it is used for aircraft design and optimization (Siggel et al., 2019, p. 367). It creates three-dimensional airplane geometries using a standardized parametric CPACS description (Siggel et al., 2019, p. 367), in which wings, fuselages, engines, control surfaces or the inner aircraft structure are described (*Features*, n.d.). TiGL offers interfaces for C, C++, Python, Java, MATLAB and FORTRAN as well as a graphical user interface (GUI) for visualizing the aircraft geometries (Siggel

et al., 2019, p. 368). The GUI, which is shipped in the program TiGL Viewer, contains a JavaScript based console that enables to automate visualizations, for example to create movies (*Features*, n.d.).

3 Method

3.1 Choice and Approach of Method

Our work aims to create a **prototype** of MBT in the context of TiGL. The lack of concrete examples on how MBT can be utilized on software systems in practice makes prototyping an appropriate approach for our work, as it allows us to practically create such a concrete example. For this purpose, we design initial test models using a selected MBT tool and execute tests on TiGL with these models. In addition, TiGL is an open-source software system (Siggel et al., 2019, p. 367) and there also exist many open-source MBT tools that we can use (Software Testing Magazine, 2022), which means that both are freely available for our work. To be able to create the best possible results, we also make use of other additional methods.

One of these additional methods is an **interview** we conduct with one of the developers of TiGL, Jan Kleinert. Since it is important to have at least some familiarity with the SUT, in our case TiGL, in order to be able to create suitable models, it is an appropriate approach to take the opportunity to speak directly with someone involved in the development of TiGL. We conduct a semi-structured interview, i.e. we have a set of fixed questions that cover TiGL in general, what is currently being tested in TiGL and specifically what could be tested in TiGL with MBT, but we also keep the option open to ask additional questions that come up during the interview. A summarized list of the questions and answers can be found in Table A1 in Appendix A.

Another additional method that we use is an **online scientific database research**. Based on the current state of research, this approach allows us to properly structure our workflow and properly define the criteria for the selection of MBT tools. We conduct a narrow forward search in Google Scholar and ResearchGate, which are both rather general, as well as in IEEE Xplore, which focuses on Computer Science. Each search term includes “Model based testing”, which can be combined with “Workflow”, “Approach” or “tool selection criteria”. If any selected paper can be refound in several databases, it could be a good sign regarding the papers and authors reliability. We deem information as relevant if it covers the workflow for MBT or the selection criteria for the MBT tools.

Furthermore, we also make use of a **web-based research** as an additional method. This is an appropriate approach as it is the easiest and fastest way to find lists of several MBT tools and then select the MBT tool we use for our work. We conduct the web search by using the search term “Model based testing tools”. We deem websites as relevant if they can provide a list of several MBT tools and a description for each MBT tool. From there, the web search continues by using the names of the MBT tools that attract our attention based on the criteria from Chapter 3.2 as individual search terms to find even more information about these MBT tools and, once an MBT tool

has been selected, to install it. The focus here lies on GitHub repositories and, if they exist, specific websites for the MBT tool.
The findings of each method are then integrated to generate the best possible prototype of MBT in the context of TiGL. Since we require the MBT tool selection criteria for our web-based research, we need to conduct the online scientific database research before we start the web-based research. In general, all additional methods need to be performed first before we start creating our prototype.

3.2 MBT Tool Selection Criteria

There exist various MBT tools, each with different characteristics (Marinescu et al., 2015, p. 91). In order to select the MBT tool that fits to our case of testing TiGL, selection criteria that cover these characteristics need to be defined. Utting et al. (2012, pp. 300-306) provide a framework for selecting MBT tools, while Marinescu et al. (2015, pp. 95-100) extend this same framework with additional dimensions. The framework we utilize is inspired by their work, however, we extend the dimension technology by the supported programming languages and add license as an additional dimension, as both aspects have not been considered in both papers. For each dimension, we define what an MBT tool must fulfill in order to be selected for our case of testing TiGL.
The dimension **technology** is considered by both Marinescu et al. (2015, pp. 95-100) and Utting et al. (2012, pp. 300-306). Our consideration of technology encompasses aspects such as the programming language compatibility, ensuring that the selected MBT tool integrates with the existing technology stack. Additionally, we scrutinize the capacity of the MBT tool for both online and offline testing, recognizing the significance of adaptability across diverse testing environments, including mobile and desktop systems. For our case, it is important that the MBT tool supports desktop applications, offline installed tools and programming languages that are supported by TiGL.
As an additional dimension that has not been considered by the current state of research, we evaluate the **license** of candidate MBT tools. The distinction between commercial and open-source solutions is pivotal for our decision-making process. We focus on open-source MBT tools, so that it is freely available for our work and, therefore, enables us to create the prototype with that MBT tool.
Marinescu et al. (2015, pp. 95-100) and Utting et al. (2012, pp. 300-306) both consider **modelling notations** as a dimension as well. The emphasis on modelling notations guides our exploration of MBT tools supporting state-based, transition-based, stochastic, and data-flow modelling. This ensures that the selected MBT tool can accommodate the intricacies of the system behaviour, allowing for a nuanced representation of states, transitions, and dynamic data interactions (Marinescu et al., 2015, pp. 96-97). We look for MBT tools that follow a state-based and transition-based approach to mapping the processes.
One of the extended dimensions of Marinescu et al. (2015, pp. 95-100) are the **artifacts**. It covers the type of information that is represented with the models created by the MBT tool (Marinescu et al., 2015, p. 97) We assess the competence of the MBT

tool in modelling and testing the functional behaviour, extra-functional aspects like performance and reliability, as well as the integration of architectural descriptions for a holistic testing approach (Marinescu et al., 2015, pp. 97-98).
Test selection criteria are another dimension that Marinescu et al. (2015, pp. 95-100) and Utting et al. (2012, pp. 300-306) both consider. These criteria define the test cases to be generated based on the requirements of the SUT (Marinescu et al., 2015, p. 98). We prioritize test selection criteria that focus on requirements-based coverage and ad-hoc test case specification. This facilitates a targeted testing approach, ensuring that our prototype of MBT generates tests that precisely correspond to requirements while allowing flexibility for ad-hoc scenarios (Utting et al., 2012, p. 304).
The **test generation method** is also considered important by Marinescu et al. (2015, pp. 95-100) and Utting et al. (2012, pp. 300-306). We assess tools for their capability in graph search algorithms, model-checking procedures, manual test case generation, and random test case generation. This diversified approach allows us to cater to different testing scenarios and project requirements (Utting et al., 2012, p. 305).
The **mapping** dimension, which only Marinescu et al. (2015, p. 100) consider, guides our assessment of tools regarding abstract and executable tests. We seek tools that provide flexibility for abstract testing in early stages, without the need for executable code, while also ensuring support for the generation and execution of directly executable tests.

4 Results

4.1 MBT Workflow

We structure the creation of the prototype in five sequential steps that are inspired by the MBT workflow outlined by Marinescu et al., (2015, pp. 93-95). However, we slightly adapt the specific steps of our MBT workflow, as we focus more on the first step of Marinescu et al., (2015, pp. 93-95), where the model is created and the tests are selected. The MBT workflow as a whole remains unchanged though.
The first of the five steps of our MBT workflow is to specify the requirements of the SUT to be tested (Marinescu et al., 2015, pp. 93-94). Here, we focus on requirements of the GUI of TiGL. The interview with one of the TiGL developers reveals that the GUI of TiGL has not been tested much. As the GUI is an important component of a software system, we decide to create a prototype of MBT that tests the GUI of TiGL.
The second and third steps of our MBT workflow are the selection of our test cases and then the formulation of these in order to create the model (Marinescu et al., 2015, pp. 93-94). Based on the specified requirements, this process results in two initial test cases. The first test case deals with correctly opening and displaying aircraft models in TiGL Viewer, the second test case deals with the correct selection and rotation of components of aircraft models in TiGL Viewer. These test cases are represented in models that are created with the selected MBT tool in Chapter 4.3.
The fourth and fifth steps of our MBT workflow deal with the mapping in the SUT TiGL with the selected test models as well as the practical execution of tests (Marinescu et al., 2015, pp. 94-95), which is covered in Chapter 4.4.

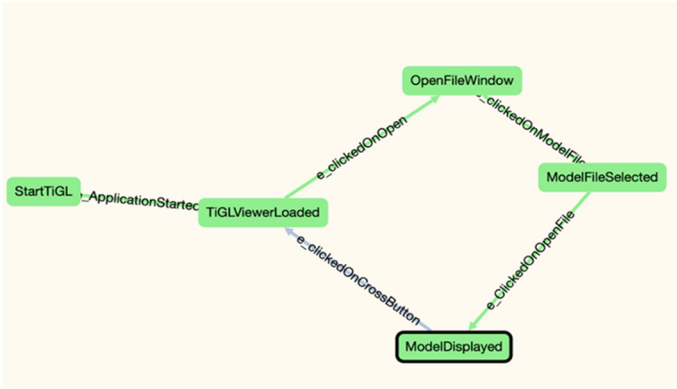
4.2 MBT Tool Selection

In order to create our prototype of MBT, we make use of an MBT tool that needs to be selected based on selection criteria that are described in Chapter 3.2. It is important that the MBT tool fits to our case of testing the GUI of TiGL.
Therefore, we select GraphWalker as the MBT tool for creating our prototype. GraphWalker is an open-source MBT tool that uses graph diagrams to model various facets of a software system in order to automatically generate test cases from this model (*GraphWalker*, n.d.). The graph diagrams are shaped in directed graphs (Software Testing Magazine, 2022), in which the nodes are system states and edges are the transitions between the states (Alégroth et al., 2022, p. 2). With the help of different variables, various aspects can be tested, including the coverage of certain nodes or edges (Olsson et al., n.d.). This MBT tool can also be used to process the functional artifacts, in our case the GUI. In addition, GraphWalker supports an offline path generation (Saket, n.d.), desktop applications (Olsson et al., n.d.) as well as the programming languages Python through the framework AltWalker (Dezmerean et al., 2023), Java and JavaScript (Olsson et al., n.d.).

4.3 Test Cases and Modelling Prototype

Here, we use GraphWalker in order to create the models that represent the two test cases that are mentioned in Chapter 4.1. The test cases are each described in prerequisites, test steps and expected results, while the models illustrate the test steps in the form of nodes and edges. The first model for correctly opening and displaying aircraft models is illustrated in Figure 1.

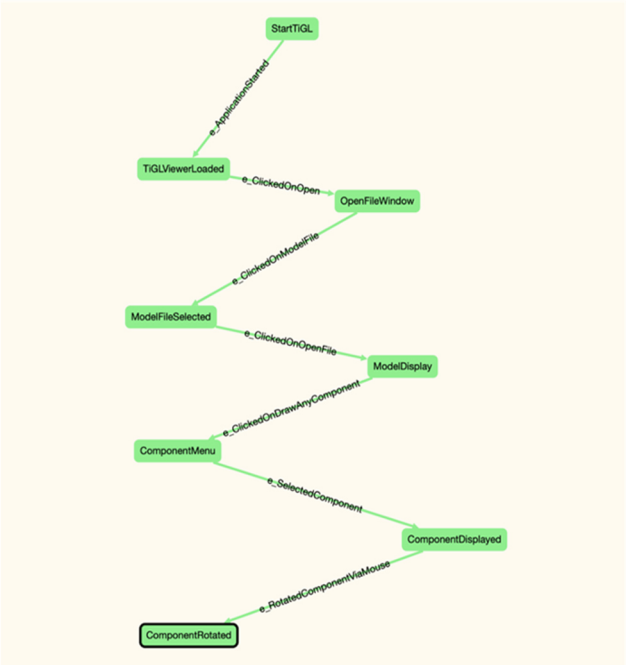
Figure 1: First GraphWalker Test Model



The prerequisites are that TiGL Viewer must be installed and there exists a valid aircraft model that can be tested. The first test case starts with launching the TiGL Viewer, by which the TiGL Viewer window should open. Then, the "Open File" button must be clicked in the menu at the top. Afterwards, a valid aircraft model needs to be selected and loaded, which then appears in TiGL Viewer. At last, TiGL Viewer is

closed. The expected outcome for the first test case is that the test steps are executed without errors, the model is successfully loaded, and TiGL is closed.
The second model for the correct selection and rotation of components of aircraft models is illustrated in Figure 2.

Figure 2: Second GraphWalker Test Model



The prerequisites are that TiGL Viewer must be installed and there exists a valid aircraft model with components that can be tested. The second test case follows the same steps as the first test case until a valid aircraft model that is selected and loaded appears in TiGL Viewer. From there on, a component of the aircraft model needs to be selected and displayed. A rotation should then be triggered through the buttons on the left side and it must be checked if the rotation is executed properly. Afterwards, the TiGL Viewer should be closed. The expected outcome for the second test case is that the test steps are executed without errors, the model and the component are successfully loaded and TiGL is closed at the end.

4.4 Running Tests

These two created test models can then be utilized for the practical implementation of the tests. Our approach involves running the tests automatically with AppleScript and use Apple Automator. With AppleScript and Automator, it is possible to control MacOS with commands and automate workflows. We write a script for each test case and execute them using the Apple Automator. In the end, both tests run successfully

and show that the two test cases function correctly. These tests can be executed multiple times. The script and workflow files are available on our GitHub repository.

5 Discussion

5.1 Contribution to Research

This work contributes to research by providing a small but concrete example on how MBT can be utilized on software systems. The workflow of applying MBT as well as the MBT tool selection criteria are both based on the current state of research. In terms of structuring the MBT workflow, this work shows that the framework inspired by Marinescu et al. (2015, pp. 93-95) works successfully in practice. It reflects a systematic and well-designed method for MBT. Furthermore, the MBT tool selection criteria that are inspired by Marinescu et al. (2015, pp. 95-100) and Utting et al. (2012, pp. 300-306) also prove useful in practice, as they cover various aspects of an MBT tool that are important for the selection. However, there are also other important aspects that have not been considered in these papers, such as the license or the supported programming languages. In general, this work adds some practical experience that, as Alégroth et al. (2022, p. 37) point out, has been largely missing in research. We also note that our work could have benefited from the experience of more practical examples by learning from mistakes of others and utilizing best practices.

5.2 Contribution to Practice

This work also contributes to practice, as this prototype can be implemented by the developers of TiGL to initiate the integration of MBT for testing TiGL. It contains not only the created prototyping models and executable tests for the GUI, but also the general MBT workflow and MBT tool selection, including the selection criteria, both of which could be used for further test cases either for the GUI or other parts of TiGL. Our work could also motivate developers of other software systems to apply MBT by adapting the MBT workflow and MBT tool selection criteria for their case, selecting an MBT tool based on their criteria, creating their own models with their MBT tool and executing the generated tests. If MBT is to be applied, we note from our experience that finding MBT tools through a web search is easy, but it requires effort and a cautious selection of criteria to choose the fitting MBT tool, which depends on the context of the system and features. It also requires effort to get used to the selected MBT tool, although some MBT tools provide examples and guidelines that can help with this.

5.3 Limitations

However, this work also contains limitations that need to be considered. First of all, the created prototype is a very simplified version of MBT that covers simple features of the GUI. This prototype should not be seen as a final product that can automatical-

ly generate and execute tests for several complex features of the GUI. In addition, our work only applies GraphWalker with the AppleScript and Apple Automator; there may exist alternative approaches for different platforms. However, further development of the prototype would have been difficult within the limited working time frame of two months and the set page limit of 10 pages, as there are also many other aspects that need to be examined under the same limitations. These include the MBT workflow and MBT tool selection based on predefined selection criteria, both of which could have been backed up with more literature. Although both aspects work in our case, considering more papers could have represented the current state of research more accurately. However, conducting a more thorough online scientific database research would also exceed the time and page limit.

5.4 Future Research

Future research should include many more concrete examples on how MBT can be utilized on software systems in practice. This could be done by creating more prototypes with different MBT tools for different types of software systems or by conducting case studies in practice. Research should also focus more on the MBT tools, especially with regard to how to optimally select an MBT tool based on given selection criteria. It might also be useful to further investigate how to optimally adapt and train developers and software testers in the use of MBT and its tools.

6 Conclusion

This work presents the creation of a prototype of applying MBT on the example of the software TiGL. For this case, the creation process is divided into four parts: the definition of an MBT workflow based on an online scientific database research, the selection of an MBT tool based on a web-based research, the creation of test models as well as the execution of the tests. Our prototype focuses on the GUI of TiGL, which, according to one of the TiGL developers, has not been tested much at the time of finalizing this work. The framework inspired by Marinescu et al. (2015, pp. 93-95) structures the MBT workflow by defining five sequential steps that are adapted to the case of TiGL: the definition of requirements for the GUI, the selection of initial test cases for the GUI, the formulation of these test cases, the mapping with TiGL as well as the practical implementation. Then, based on selection criteria that are inspired by Marinescu et al. (2015, pp. 95-100) and Utting et al. (2012, pp. 300-306) and that are adapted to the case of TiGL, GraphWalker, an open-source MBT tool that automatically generates test cases from graph models (*GraphWalker*, n.d.), has been selected for our prototype. Furthermore, two test cases have been selected and modelled with GraphWalker. The first test case deals with correctly opening and displaying aircraft models in TiGL Viewer, the second test case deals with the correct selection and rotation of components of aircraft models in TiGL Viewer. With the created models that represent these two test cases, the tests are then automatically executed with AppleScript and Apple Automator, leading to successful test results.

References

1. Alégroth, E., Karl, K., Rosshagen, H. et al. Practitioners' best practices to Adopt, Use or Abandon Model-based Testing with Graphical models for Software-intensive Systems. Empir Software Eng 27, 103 (2022). https://doi.org/10.1007/s10664-022-10145-2
2. Dezmerean R., et al. (2023). AltWalker. GitHub. https://github.com/altwalker/altwalker
3. *Features* n.d., *TiGL*, accessed 23 January 2024, https://dlr-sc.github.io/tigl/pages/features.html
4. *GraphWalker* n.d., accessed 30 January 2024, https://graphwalker.github.io/
5. Marinescu, R. & Seceleanu, C. & Guen, H. & Pettersson, P. (2015). A Research Overview of Tool-Supported Model-based Testing of Requirements-based Designs. 10.1016/bs.adcom.2015.03.003.
6. *Model Based Testing* 2020, *ProfessionalQA*, accessed 30 January 2024, https://www.professionalqa.com/model-based-testing-tools
7. Olsson N., et al. (n.d.). GraphWalker. GitHub. https://github.com/GraphWalker
8. Saket n.d., *automated-360*, accessed 30 January 2024, https://automated-360.com/model-based-testing/model-based-testing/
9. Schieferdecker I., "Model-Based Testing," in IEEE Software, vol. 29, no. 1, pp. 14-18, Jan.-Feb. 2012, doi: 10.1109/MS.2012.13.
10. Siggel M., Kleinert J., Stollenwerk T. et al.: TiGL: An Open Source Computational Geometry Library for Parametric Aircraft Design, Math.Comput.Sci. (2019). https://doi.org/10.1007/s11786-019-00401-y
11. Software Testing Magazine 2022, *Software Testing Magazine*, accessed 30 January 2024, https://www.softwaretestingmagazine.com/tools/open-source-model-based-testing-tools/
12. Utting M., Pretschner A., Legeard B. (2012) A taxonomy of model-based testing approaches. Software testing, verification and reliability 22(5):297–312

A. Appendix

Table A1: Summarized Questions and Answers of Interview, conducted online on 21 December 2023

Questions (Kaan Akbulut and Johannes Simon)	**Answers (Jan Kleinert)**
What parts does TIGL consist of?	TiGL consists of a modeling library and the TiGL Viewer.
Is MBT currently being used for testing TiGL?	No, MBT has not yet been used at all for testing TiGL.
What could be tested with MBT in TiGL (in terms of features)?	Currently, many Python files are not being tested. There is also a lack of tests for the GUI. Both can be options for the initial use of MBT.
Do you have one specific example of GUI Testing?	You could test basic functions of TiGL Viewer, like loading Files etc.