ISBN 978 -987-633-574-4

Expanding the scope of a testing framework for Industry 4.0

Martín L. Larrea[1,2,3] and Dana K. Urribarri[1,2,3]

[1] Department of Computer Science and Engineering, Universidad Nacional del Sur (UNS), Bahía Blanca, Argentina
[2] Computer Graphics and Visualization R&D Laboratory, Universidad Nacional del Sur (UNS) - CIC Prov. Buenos Aires, Bahía Blanca, Argentina
[3] Institute for Computer Science and Engineering, Universidad Nacional del Sur (UNS) - CONICET, Bahía Blanca, Argentina
{mll, dku}@cs.uns.edu.ar

Abstract. Software has become a transversal and foundational element of the new industrial revolution, with a growing presence in every stage of production processes. Software inclusion has grown not only quantitative but also qualitative; this increase turns critical the impact of software on society, the environment, and individuals. This new era requires new methodologies, techniques, and tools to verify and validate the foundations of Industry 4.0. This paper aims to expand the capacity of a current testing framework to be more flexible and require less specialized personnel. We introduced a web application that complements it through the generation of test cases. The framework and web application are open sources and freely available which means that these software elements are a foundation that will allow future development teams to continue expanding their functionality and application areas within Industry 4.0.

Keywords: Industry 4.0, Verification and Validation, Testing, Message Sequence Specification, Aspect-Oriented Programming, Software

1 Introduction

Software quality has become one of the most important factors in determining the success of products or companies in the era of Industry 4.0. As stated by Oztemel et. al [9] "...the superior quality of the manufacturing industry strictly depends on its high-quality applied production technology..." and "...there are now companies having the largest part of businesses in their sector with only running a software...". Lee et. al [7] also highlighted the key role of software in this new industry: "The Fourth Industrial Revolution is ubiquitous and will increasingly transform and reshape operations/production, supply-chain, management, and governance as well as products and services. Whatever could be codified of the organizational life will be put into codes and software and embedded into cybernetics systems that will replace human work activities". Yang [8], as well, described that "Industry 4.0 has two key factors: integration and

ISBN 978 -987-633-574-4

interoperability. Integrated with applications and software systems, Industry 4.0 will achieve seamless operations across organizational boundaries and will realize networked organizations". To a great extent, the success of Industry 4.0 rests on the quality of the software, which has the responsibility to ensure that it is error-free.

In 2020 we present the latest version of our testing framework called TAPIR [5], which was designed to detect failures in the sequence of method calls. It was suitable for Industries 4.0, but it was only available as a tool for Java developers. In this paper, we expand the scope of the framework by adding a new component: a web application that helps the software developer or other professionals to generate test cases to test their software element regardless of the programming language. The web application was implemented using TypeScript and React. Keeping the TAPIR philosophy, all the development presented in this work is free and open source.

The rest of the paper is structured as follows. Section 2 provides background information about the TAPIR framework. Then, we present the contributions of this article in Section 3. We later show how it was possible to find an error using the web application. Finally, we conclude with a brief discussion on the limitations and advantages of our approach and the future work.

2 TAPIR

TAPIR [5] is a testing framework for object-oriented source code based on Message Sequence Specification (MSS) [3] and implemented using Aspect-Oriented Programming (AOP) [4]. AOP allows the framework to create test cases that execute automatically with each execution of the program under test without modifying its source code. The use of MSS allows the developer to describe a regular expression for each class, which represents its correct behavior. The framework executes the program, and it checks whether the methods are invoked according to the regular expression in the class specification. The first thing the developer must do to use the framework is to create the regular expressions associated with the classes under test. These regular expressions must specify the correct behavior or invocation order of the classes' methods. To simplify the regular expression writing, symbols (i.e. characters) are used instead of the actual names of the methods. However, to be able to interpret it, the developer must specify a mapping between the actual methods' names and their corresponding symbol. The regular expressions and the maps between methods and symbols are set in the *TestingSetup.java* class.

The framework consists of two main components: an aspect, and a java class. The aspect is named *TestingCore.aj* and it contains the implementation of the framework's core. Listing 1.1 shows the implementation of the *TestingSetup.java* class that describes to TAPIR the correct behavior of example classes *CA* and *CB*. In this case, the correct usage of class *CA* states that, after the creation of the object, there should be a call to f followed by a call to g. After that, there can be as many calls as desired to either g or h. The final call of the sequence must

ISBN 978 -987-633-574-4

be to *h*. To correct use class *CB*, there should be first a call to *alpha* followed by a call to *gamma*, or a call to *gamma* followed by a call to *beta*. Afterward, any method between *alpha*, *beta*, or *gamma* can be called.

Listing 1.1: TAPIR configuration for classes *CA* and *CB*

```
//Class CA: Definition of the methods and their corresponding symbols
mapObjectsToCallSequence = new HashMap<Integer, String>();
mapMethodsToSymbols = new HashMap<String, String>();
mapMethodsToSymbols.put("main.CA.<init>", "c");
mapMethodsToSymbols.put("main.CA.f", "f");
mapMethodsToSymbols.put("main.CA.g", "g");
mapMethodsToSymbols.put("main.CA.h", "h");
//Definition of the regular expression
regularExpression = Pattern.compile("cfg(g|h)*h");
//Initializing the regular expressions controller
matcher = regularExpression.matcher("");
//A TestingInformation instance stores all information related to how the class is tested
TestingInformation ti = new TestingInformation(CA.class.toString(),
    mapObjectsToCallSequence, mapMethodsToSymbols, regularExpression, matcher,
    true);
TestingCore.mapClassToTestingInformation.put(CA.class.toString(), ti);
//Class CB: Definition of the methods and their corresponding symbols
mapObjectsToCallSequence = new HashMap<Integer, String>();
mapMethodsToSymbols = new HashMap<String, String>();
mapMethodsToSymbols.put("main.CB.alpha", "a");
mapMethodsToSymbols.put("main.CB.gamma", "g");
mapMethodsToSymbols.put("main.CB.beta", "b");
//Definition of the regular expression
regularExpression = Pattern.compile("(ag|gb)(a|g|b)*");
//Initializing the regular expressions controller
matcher = regularExpression.matcher("");
//A TestingInformation instance stores all information related to how the class is tested
ti = new TestingInformation(CB.class.toString(), mapObjectsToCallSequence,
    mapMethodsToSymbols, regularExpression, matcher, false);
TestingCore.mapClassToTestingInformation.put(CB.class.toString(), ti);
```

In Listing 1.3, we can see the framework output when the code portion of Listing 1.2 corresponding to the *CA* class is executed. In this case, the last call to *f* does not follow the MSS specified for the *CA* class. As mentioned above, when an error is detected, TAPIR informs by console the class and object that produced the error. The method that violated the MSS, the MSS and the actual sequence of calls are also shown in the console. Finally, the system aborts the execution, as indicated by the last parameter of method *TestingInformation* in the configuration.

Listing 1.2: Two snippets of code showing examples of wrong usage of class *CA* and class *CB*.

```
CA ca1 = new CA();                    ca1.f();
```

ISBN 978 -987-633-574-4

```
ca1.g();                          cb1.alpha();
ca1.h();                          cb1.alpha();
ca1.f();                          cb1.gamma();
CB cb1 = new CB();                cb1.gamma();
```

Listing 1.3: Error example for the CA class. The execution is aborted when the error is found.

```
---    ERROR FOUND   ---
Class: class main.CA
Object Code: 977993101
Method Executed: main.CA.f
Regular Expression: cfg(g|h)*h
Execution Sequence: cfghf
----- SYSTEM ABORTING... -----
```

Listing 1.4: Error example for the CB class. The execution is allowed to continue when the error is found.

```
---    ERROR FOUND   ---              ---    ERROR FOUND   ---
Class: class main.CB                  Class: class main.CB
Object Code: 859417998                Object Code: 859417998
Method Executed: main.CB.alpha        Method Executed: main.CB.gamma
Regular Expression: (ag|gb)(a|g|b)*   Regular Expression: (ag|gb)(a|g|b)*
Execution Sequence: aa                Execution Sequence: aag
-- CONTINUING EXECUTION... ---        -- CONTINUING EXECUTION... ---
```

Listing 1.4 shows the framework output when the code portion of Listing 1.2 corresponding to the class *CB* is executed. In this case, the second call to *alpha* does not follow the MSS specification for class *CB*. As configured in Listing 1.2 , the last parameter in the call to method *TestingInformation* is false, indicating that the execution must continue despite the existing errors. Therefore, Listing 1.4 shows multiple errors.

For a more in-depth analysis of the framework and more usage examples, we recommend that the reader see [5].

3 Proposal

As we previously mentioned, TAPIR is developed in Java and can only be used in Java applications. Although programming knowledge is necessary, the framework can be used by developers who do not have specific knowledge of the testing area. These two restrictions present two opportunities for improvement, and these are the contribution proposal for this work.

The proposal in this work is to expand the development carried out in [5] to include a new piece of software, a web application that completes the technique previously presented. While the framework functionality is oriented to evaluate

ISBN 978 -987-633-574-4

the correct usage of a set of running classes, the web application allows the generation of test cases to test more methodically the behavior of a class against possible combinations of calls of its methods. This is useful for testing software components that are not yet part of a complete application. In this way, the web application generates documentation oriented to the unit testing of such components. This documentation is of great help to the developer and, since the web application is very easy to use, any member of the work team can generate the test cases.

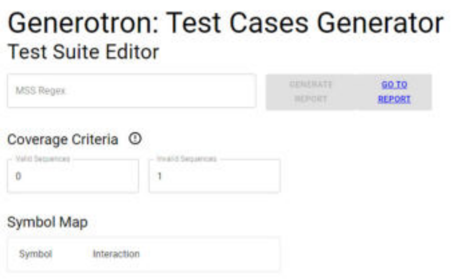
The definition, design, and implementation of the application and its interactions with the user were conceived under a User-Centered Design strategy [2] and considering the work by Signoretti et. al [10]. Under this strategy, the user of a system has active participation in its design and development. In our case, we worked with users who did not have a computer profile but were familiar with the software industry and Industry 4.0 in general. With them, the design of the graphical interface of the system, the use of labels throughout the application, and the interactions were validated. Interdisciplinary teams [11] are an increasingly common way of working in the context of Industries 4.0, so the tools that ensure the quality of software products should be usable by all team members. In this way, it would be possible to test a greater part of the software or test the same as before but in less time. On the other hand, making the testing tool independent from the language and even from the software and hardware platforms would allow the same tool to be applied in different projects, reducing or eliminating training times in new methodologies or programs.

3.1 Implementation

To avoid the MSS parsing logic being tied to the web application, the project was divided into two parts which are the MSS-Parser and the MSS-App modules. The MSS-Parser module validates that text strings are correct MSS and generates the test cases according to the provided coverage parameters. This module was not implemented from scratch but was a fork of the genex.js project repository, authored by Alix Axel. MSS-Parser was implemented entirely in TypeScript, to make it easier to use by providing a statically typed API. Like genex.js, MSS-parser uses the ret (Regular Expression Tokenizer) library to parse the regular expression associated with the MSS and return a tree of tokens. Then, this tree is traversed to generate the strings that represent test cases. The MSS-App was developed using React as the front-end framework, also in TypeScript, to be consistent with the MSS-Parser module and take advantage of static typing. The Material-UI web-component library was used because it offers a wide variety of components developed following the Material Design standards [1].

3.2 The front-end design

The front-end is divided into two parts. On the one hand, it offers an editor (Figure 1a) that allows entering the MSS, the coverage parameters, and an optional mapping between symbols and interaction names. Once the values are entered and validated (Figure 1b), the application generates the test cases and enables the second part of the application. There the user can visualize the generated report (Figure 2a) and, for each test case the user can indicate if each step could be executed, and also if the test was successful or not. Each test case can contain a text note.



Generotron: Test Cases Generator

Test Suite Editor

MSS Regex

Coverage Criteria

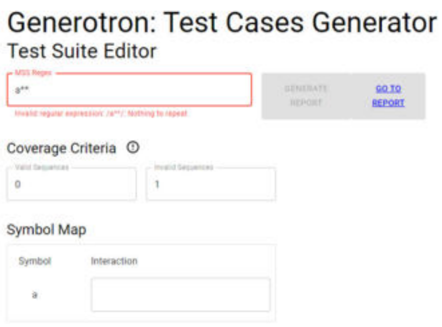
Valid Sequences: 0

Invalid Sequences: 1

Symbol Map

Symbol | Interaction

(a) Homescreen of Generotron, a web application for the generation of test cases based on MSS.



Generotron: Test Cases Generator

Test Suite Editor

MSS Regex: a**

Invalid regular expression: /a**/: Nothing to repeat

Coverage Criteria

Valid Sequences: 0

Invalid Sequences: 1

Symbol Map

Symbol | Interaction

a

(b) Error found in the MSS input field. Every time the user types something in this field, it is checked

Fig. 1: Generotron: Test Case Generator

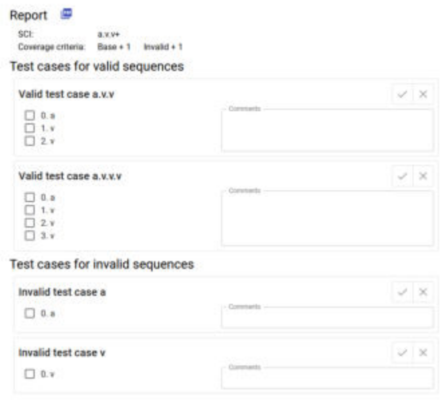
The application offers an extremely simple editor. Figure 1a shows a screenshot in its initial state. As seen in the figure, it is composed of three mandatory input fields in which the user enters the regular expression and the two values used as parameters for the coverage criteria. These last two are initialized by default with the values 0 and 1 since they are the minimum values allowed by definition. In turn, the input fields do not allow entering smaller values. Once a valid MSS expression has been entered, new fields are dynamically generated in which the user can enter the full name of the interaction, as shown in Figure 3.

Symbol mapping is optional at the individual level; the user can add a more descriptive name for a symbol while omitting those where it is considered unnecessary. When viewing the report, the names added to the mapping are used to display more descriptive versions of the MSS expression and the list of methods to execute in each test case. It is important to note that although the mapping is optional, whenever abbreviated symbols are used, the ideal would be to provide one to improve the readability of the generated report.

In case an error is detected in the entered values, a message is displayed with a description (Figure 1b). The MSS-Parser module provides these error messages. Once the required values are entered, the Generate Report button is enabled. The web report was designed and implemented prioritizing simplicity over fanciness so that the same web format presented in the browser could be exported to a PDF using directly the universally provided printing functionality. At the top, it has a heading like the one in Figure 3, which shows the values

ISBN 978 -987-633-574-4

previously entered in the editor and used to generate the test cases in the report. The blue PDF icon is a button that allows exporting the document as a PDF file. Then the corresponding test cases are listed and grouped according to whether they are valid or invalid sequences of interactions. Each test case is contained by a box like the one shown in Figure 2a. The title includes the sequence of interactions from the MSS expression that compose the test case. As mentioned above, when a mapping was provided, the list of methods includes names instead of symbols as shown in Figure 6. The report includes a checkbox on each method of the test cases to indicate a successful execution. After conducting the test case, the user can register a successful or failed result in the upper right corner and write comments if necessary. A valid test case is successful if all the methods were successful. However, an invalid test case is successful if it fails at some point.



Report

SCI: a.v.v+
Coverage criteria: Base + 1 Invalid + 1

Test cases for valid sequences

Valid test case a.v.v
- ☐ 0. a
- ☐ 1. v
- ☐ 2. v

Comments

Valid test case a.v.v.v
- ☐ 0. a
- ☐ 1. v
- ☐ 2. v
- ☐ 3. v

Comments

Test cases for invalid sequences

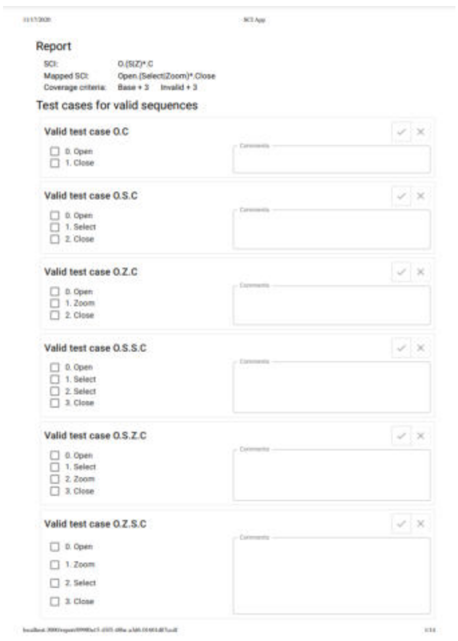
Invalid test case a
- ☐ 0. a

Comments

Invalid test case v
- ☐ 0. v

Comments

(a) A webform that shows the generated test cases. For each one of them, the user can indicate if each step could be executed, and also if the test was successful or not. Each test case can contain a text note.



SCI App

Report

SCI: O.(S|Z)*.C
Mapped SCI: Open.(Select|Zoom)*.Close
Coverage criteria: Base + 3 Invalid + 3

Test cases for valid sequences

Valid test case O.C
- ☐ 0. Open
- ☐ 1. Close

Comments

Valid test case O.S.C
- ☐ 0. Open
- ☐ 1. Select
- ☐ 2. Close

Comments

Valid test case O.Z.C
- ☐ 0. Open
- ☐ 1. Zoom
- ☐ 2. Close

Comments

Valid test case O.S.S.C
- ☐ 0. Open
- ☐ 1. Select
- ☐ 2. Select
- ☐ 3. Close

Comments

Valid test case O.S.Z.C
- ☐ 0. Open
- ☐ 1. Select
- ☐ 2. Zoom
- ☐ 3. Close

Comments

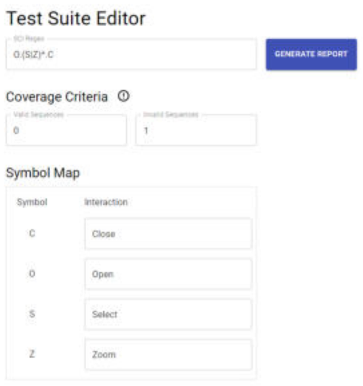
Valid test case O.Z.S.C
- ☐ 0. Open
- ☐ 1. Zoom
- ☐ 2. Select
- ☐ 3. Close

Comments

(b) The webform with the test cases can be easily exported to PDF.

Fig. 2: Report generation in Generotron

ISBN 978 -987-633-574-4



Test Suite Editor

SCI Regex: O.(S|Z)*.C

GENERATE REPORT

Coverage Criteria

Valid Sequences: 0

Invalid Sequences: 1

Symbol Map

Symbol	Interaction
C	Close
O	Open
S	Select
Z	Zoom

Fig. 3: For each symbol that is used in the MSS, it is possible to establish a mapping with the method it represents. This makes it easier to read the test cases.

4 Test Case. Rock.AR, a software solution for point counting

Point counting is the standard method to establish the modal proportion of minerals in coarse-grained igneous, metamorphic and sedimentary rock samples. This method requires taking observations at regular positions on the sample, namely grid intersections. Rock.AR [6] is an open-source visualization tool developed in Java that implements a semiautomatic point-counting method.

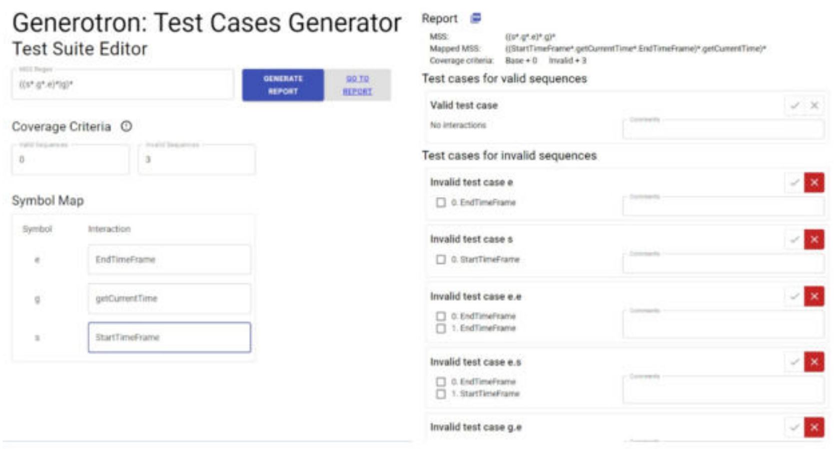
The implementation of Rock.AR includes a class called *CurrentTime* that provides the current time and is also used to measure the elapsed time between two moments. This class is part of a utility package that the application uses. There are only three methods available. *GetCurrentTime* returns the current time, conforming to format yyyy-MM-dd HH:mm:ss. *StartTimeFrame* is used to mark the beginning of a time frame, and *EndTimeFrame* marks the end of such time frame and returns the elapsed time between start and end.

The correct use of this class is described as follows: *GetCurrentTime* can be called at any time, and a time interval can be measured by first calling *StartTimeFrame* and then *EndTimeFrame*. Between a call to *StartTimeFrame* and *EndTimeFrame*, calls to *getCurrentTime* can occur. Using these symbols for each method; g for *GetCurrentTime*, s for *StartTimeFrame*, and e for *EndTimeFrame*, the following MSS describes the correct operation of the class:

$$((s^* \bullet g^* \bullet e)^* | g)^* \tag{1}$$

Using the Web Application to generate test cases for the *CurrentTime* class (see Figure 4), it was possible to detect a bug. The application generated combinations of calls to the methods of the class that caused the class to behave

ISBN 978 -987-633-574-4


Generotron: Test Cases Generator
Test Suite Editor
MSS Regex: ((s*.g*.e)*|g)*
GENERATE REPORT
GO TO REPORT
Coverage Criteria
Valid Sequences: 0
Invalid Sequences: 3
Symbol Map

Symbol	Interaction
e	EndTimeFrame
g	getCurrentTime
s	StartTimeFrame

Report
MSS: ((s*.g*.e)*.g)*
Mapped MSS: ((StartTimeFrame*.getCurrentTime*.EndTimeFrame)*.getCurrentTime)*
Coverage criteria: Base + 0 Invalid + 3
Test cases for valid sequences
Valid test case
No interactions
Test cases for invalid sequences
Invalid test case e
☐ 0. EndTimeFrame
Invalid test case s
☐ 0. StartTimeFrame
Invalid test case e.e
☐ 0. EndTimeFrame
☐ 1. EndTimeFrame
Invalid test case e.s
☐ 0. EndTimeFrame
☐ 1. StartTimeFrame
Invalid test case g.e

Fig. 4: Test cases generated for the *GetCurrentTime* class

incorrectly when tested. Although these sequences should not occur during normal execution, the class should be robust enough to withstand misuse, which is not. Note that a valid test case is successful if the sequence executes correctly; however, an invalid test case is successful if the sequence fails to execute.

5 Conclusions & Future Work

Industry 4.0 requires new methodologies to ensure the quality of its software, a key element in its production chain. A framework for testing Object-Oriented Software was developed for testing Java applications but was only available for this programming language and could only be used by someone with programming knowledge. We expanded this framework by introducing a web application that complements it. Although the framework can be used in any Java implementation without modifying the source code, it requires Java and programming knowledge. The web application is suitable for any implementation based on Object-Oriented Programming, regardless of the programming language and it can be used by anyone in the work team. All these tools were designed and implemented to detect, without modifying the source code, failures in the sequence of calls that objects make. As shown in the case studies, these tools help with the detection of errors that would otherwise be difficult to find. The framework is available for downloading at `http://cs.uns.edu.ar/~mll/lapaz/` and the web application can be used at `https://cs.uns.edu.ar/~dku/mss`. The source code is available and licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

ISBN 978 -987-633-574-4

Acknowledgment

This work was partially supported by the following research projects: PGI 24/N050 and PGI 24/ZN35 from the Secretaría General de Ciencia y Tecnología, Universidad Nacional del Sur, Argentina.

References

1. Ian G Clifton. *Android user interface design: Implementing material design for developers.* Addison-Wesley Professional, 2015.
2. Roger Coleman, John Clarkson, and Julia Cassim. *Design for inclusivity: A practical guide to accessible, innovative and user-centred design.* CRC Press, 2016.
3. Shekhar Kirani and W. T. Tsai. Specification and verification of object-oriented programs. Technical report, Computer Science Department, University of Minnesota, 1994.
4. Ramnivas Laddad. *AspectJ in Action: Practical Aspect-Oriented Programming.* Manning Publications Co., Greenwich, CT, USA, 2003.
5. Martin Larrea and Dana Urribarri. Increasing confidence in industry 4.0 through new software verification and validation techniques. In *International Conference on Production Research ICPR Americas*, page In press. International Conference of Production Research, 2020.
6. Martín L Larrea, Silvia M Castro, and Ernesto A Bjerg. A software solution for point counting. petrographic thin section analysis as a case study. *Arabian Journal of Geosciences*, 7(8):2981–2989, 2014. doi:10.1007/s12517-013-1032-0.
7. MinHwa Lee, JinHyo Joseph Yun, Andreas Pyka, DongKyu Won, Fumio Kodama, Giovanni Schiuma, HangSik Park, Jeonghwan Jeon, KyungBae Park, KwangHo Jung, et al. How to respond to the fourth industrial revolution, or the second information technology revolution? dynamic new combinations between technology, market, and society through open innovation. *Journal of Open Innovation: Technology, Market, and Complexity*, 4(3):21, 2018.
8. Yang Lu. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of industrial information integration*, 6:1–10, 2017.
9. Ercan Oztemel and Samet Gursev. Literature review of industry 4.0 and related technologies. *Journal of Intelligent Manufacturing*, 31(1):127–182, 2020.
10. Ingrid Signoretti, Larissa Salerno, Sabrina Marczak, and Ricardo Bastos. Combining user-centered design and lean startup with agile software development: a case study of two agile teams. In *International Conference on Agile Software Development*, pages 39–55. Springer, 2020.
11. Alp Ustundag and Emre Cevikcan. *Industry 4.0: managing the digital transformation.* Springer, 2017.