|  |  |
| --- | --- |
| **agh_znk_wbr_rgb_150ppi** | **AKADEMIA GÓRNICZO-HUTNICZA**  **im. Stanisława Staszica w Krakowie**  **WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI** |

**Praca dyplomowa**

**inżynierska**

|  |
| --- |
| **Andrzej Szewczyk** |
| *Imię i nazwisko* |
| **Inżynieria Mechatroniczna** |
| *Kierunek studiów* |
| **Uniwersalny symulator slave node’ów dla sieci CANopen w LabVIEW.** |

*Temat pracy dyplomowej*

|  |
| --- |
| **Universal CANopen slave nodes simulator in LabVIEW.** |
| *Subject of engineer diploma thesis* |

|  |  |  |
| --- | --- | --- |
| **dr hab. inż. Tomasz Barszcz, prof. AGH** |  | ………………….. |
| *Promotor pracy* |  | *Ocena* |

Kraków, rok 20...../20.....

Kraków, ……………..

|  |  |
| --- | --- |
| Imię i nazwisko: | Andrzej Szewczyk |
| Nr albumu: | 270039 |
| Kierunek studiów: | Inżynieria Mechatroniczna |
| Specjalność: |  |

**OŚWIADCZENIE**

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tj. Dz.U.z 2006 r. Nr 90, poz. 631 z późn.zm.) : „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust.1 ustawy z dnia 27 lip[ca 2005 r. Prawo o szkolnictwie wyższym (tj. Dz.U. z 2012 r. poz. 572, z późn.zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem (-am) ze źródeł innych niż wymienione w pracy”.

.....................................................

*podpis dyplomanta*

Kraków, ……………..

|  |  |
| --- | --- |
| Imię i nazwisko: | Andrzej Szewczyk |
| Nr albumu: | 270039 |
| Kierunek studiów: | Inżynieria Mechatroniczna |
| Specjalność: |  |

**OŚWIADCZENIE**

Świadomy/a odpowiedzialności karnej za poświadczanie nieprawdy oświadczam, że niniejszą inżynierską pracę dyplomową wykonałem/łam osobiście i samodzielnie oraz nie korzystałem/łam ze źródeł innych niż wymienione w pracy.

Jednocześnie oświadczam, że dokumentacja praca nie narusza praw autorskich   
w rozumieniu ustawy z dnia 4 lutego 1994 roku o prawie autorskim i prawach pokrewnych (Dz. U. z 2006 r. Nr 90 poz. 631 z późniejszymi zmianami) oraz dóbr osobistych chronionych prawem cywilnym. Nie zawiera ona również danych i informacji, które uzyskałem/łam w sposób niedozwolony. Wersja dokumentacji dołączona przeze mnie na nośniku elektronicznym jest w pełni zgodna z wydrukiem przedstawionym do recenzji.

Zaświadczam także, że niniejsza inżynierska praca dyplomowa nie była wcześniej podstawą żadnej innej urzędowej procedury związanej z nadawaniem dyplomów wyższej uczelni lub tytułów zawodowych.

………………………………..

*podpis dyplomanta*

Kraków, ……………..

Imię i nazwisko: Andrzej Szewczyk

Adres korespondencyjny: 42-100 Kłobuck, ul. Kościuszki 37

Temat pracy dyplomowej inżynierskiej: Uniwersalny symulator slave node’ów dla sieci CANopen w LabVIEW.

Subject of engineer diploma thesis: Universal CANopen slave nodes simulator in LabVIEW.

Rok ukończenia: 2017

Nr albumu: 270039

Kierunek studiów: Inżynieria Mechatroniczna

Profil dyplomowania:

**OŚWIADCZENIE**

Niniejszym oświadczam, że zachowując moje prawa autorskie, udzielam Akademii Górniczo-Hutniczej im. S. Staszica w Krakowie nieograniczonej w czasie nieodpłatnej licencji niewyłącznej do korzystania z przedstawionej dokumentacji inżynierskiej pracy dyplomowej, w zakresie publicznego udostępniania i rozpowszechniania w wersji drukowanej i elektronicznej.

Kraków, ...............… ……………………………..

*data podpis dyplomanta*

i Na podstawie Ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (Dz.U. 2005 nr 164 poz. 1365) Art. 239. oraz Ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz.U. z 2000 r. Nr 80, poz. 904, z późn. zm.) Art. 15a. "Uczelni w rozumieniu przepisów o szkolnictwie wyższym przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli uczelnia nie opublikowała pracy dyplomowej w ciągu 6 miesięcy od jej obrony, student, który ją przygotował, może ją opublikować, chyba że praca dyplomowa jest częścią utworu zbiorowego."

Kraków, ……………..

**AKADEMIA GÓRNICZO-HUTNICZA**

**WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI**

**TEMATYKA PRACY DYPLOMOWEJ INŻYNIERSKIEJ**

dla studenta IV roku studiów stacjonarnych

*imię i nazwisko studenta*

|  |  |
| --- | --- |
| TEMAT PRACY DYPLOMOWEJ IINŻYNIERSKIEJ: |  |

Uniwersalny symulator slave node’ów dla sieci CANopen w LabVIEW. SUBJECT OF ENGINEER DIPLOMA THESIS:

Universal CANopen slave nodes simulator in LabVIEW.

*Promotor pracy:* dr hab. inż. Tomasz Barszcz, prof. AGH

*Recenzent pracy:* dr. inż. Adam Jabłoński *Podpis dziekana:*

PLAN PRACY DYPLOMOWEJ

1. Omówienie tematu pracy i sposobu realizacji z promotorem.
2. Zebranie i opracowanie literatury dotyczącej tematu pracy.
3. Zebranie i opracowanie wyników badań.
4. Analiza wyników badań, ich omówienie i zatwierdzenie przez promotora.
5. Opracowanie redakcyjne.

Kraków, ...............… ……………………………..

*data podpis dyplomanta*

**TERMIN ODDANIA DO DZIEKANATU: 20        r.**

*podpis promotora*

Kraków, ……………..

Akademia Górniczo-Hutnicza im. Stanisława Staszica

**Wydział Inżynierii Mechanicznej i Robotyki**

Kierunek: Inżynieria Mechatroniczna

Profil dyplomowania:

**Andrzej Szewczyk**

**Praca dyplomowa inżynierska**

**Uniwersalny symulator slave node’ów dla sieci CANopen w LabVIEW.**

Opiekun: dr hab. inż. Tomasz Barszcz, prof. AGH

STRESZCZENIE

W ostatnich latach miała miejsce gigantyczna digitalizacja życia codziennego każdego z nas. Zastanówmy się ile jest przedmiotów w zasięgu naszego wzroku, w których znajduje się mikroprocesor realizujący pewien ciąg instrukcji na podstawie kodu. Osoby niezwiązane w branżą IT zapewne dojdą do następującego wniosku: skoro urządzenia lub aplikacje, które są na co dzień używane, wymagają dobrze napisanego kodu w celu spełnienia oczekiwań użytkownika, to zapewne potrzeba skończonej liczby programistów, którzy ten kod napiszą. Niestety, nic bardziej mylnego. Aby firma tworząca oprogramowanie mogła odnieść komercyjny sukces, musi ona, oprócz wykwalifikowanych pracowników, posiadać również skuteczny proces, który zapewni, że oprogramowanie dostarczone końcowemu klientowi jest możliwie wysokiej jakości.

Celem niniejszej pracy magisterskiej jest zaprezentowanie narzędzia, które prowadzi nie tylko do zwiększenia jakości końcowego kodu, ale również zmniejsza ryzyko niepowodzenia projektu oraz przyspiesza czas jego realizacji. Systemy ciągłej integracji oparte na regularnym dostarczaniu, budowaniu i automatycznym testowaniu zintegrowanych wersji kodu stały się normą w nowoczesnych firmach programistycznych. Rozwiązanie opisane w tej pracy może dodatkowo usprawnić proces ciągłej integracji, ponieważ natychmiastowe wykonanie optymalnego zestawu testów regresyjnych dobranych na podstawie zmian w kodzie, zapewnia wartościową informację na temat jakości testowanej wersji oprogramowania. Zaprezentowane narzędzie mogłoby być skutecznie wykorzystane w fazie rozwoju dużych systemów informatycznych oraz w międzynarodowych, dużych zespołach programistycznych.

Kraków, the……………..

AGH University of Science and Technology

**Faculty of Mechanical Engineering and Robotics**

Field of Study: Mechatronics engineering

Specializations:

**Andrzej Szewczyk**

**Engineer Diploma Thesis**

**Universal CANopen slave nodes simulator in LabVIEW.**

Supervisor: dr hab. inż. Tomasz Barszcz, prof. AGH

SUMMARY

In the recent years, reality of everyday life has undergone the real digital transformation. Let’s think about how many devices in our sight have a CPU[[1]](#footnote-1), which is executing a set of instructions based on the code. Most people that are not familiar with the IT[[2]](#footnote-2) industry will draw the following conclusion: since all devices or applications that are used every day require a well-written code in order to satisfy end-user needs, there is a need to have a finite number of programmers to write this code. Still, there is nothing more wrong. If a company that delivers software wants to achieve the success, besides good programmers, this company must have a process, which is aimed at ensuring that the software delivered to an end-customer has the best possible quality.

The purpose of this master thesis is to propose a tool that leads not only to   
quality improvement of the final code, but also mitigates the risks in a project and accelerates time to market. Continuous integration systems that have been built based on regular delivery of automatically pre-tested integrated code builds are a common practice in modern IT companies. The solution described in this thesis could further enhance the continuous integration process, because immediate execution of an optimal regression test suite, which is selected based on changes in the latest code commit, provides valuable information on quality of the software version under test.   
The proposed tool may be effectively used in the development phase of large-scale IT systems and in a big international team of programmers.

# Introduction

This chapter is intended to introduce the reader to the content of the thesis. It comprises four sections. Section 1.1. defines the main goal of the thesis and illustrates the steps taken to achieve it. Section 1.2. briefly presents the role of continuous integration in modern software development methodologies. Section 1.3. discusses how my interest in the CI[[3]](#footnote-3) process has aroused, including a mention of two years of my professional experience with the software development process in the multiple international teams. Section 1.4 examines sources of information which refer to the subject of the thesis.

## Goal and plan of the thesis

TBD

## Why continuous integration is important

Agile methodologies are currently one of the most well-known and frequently used software development life cycle models. It is difficult to find a job offer for a programmer that does not contain any of the following words: Agile, Scrum, pair programming, CI, CD[[4]](#footnote-4) and much more associated with the agile methodologies.

In a nutshell, there are two main goals of CI: to automatically generate a software build and to provide developers with immediate feedback about quality of the recent code build. This approach to the software development process claims to be more human friendly than traditional development methods.

At this place, I would like to make a reference to Manifesto for Agile Software Development. This manifesto was proclaimed by the seventeen signatories during a meeting in Snowbird, Utah between 11th and 13th of February 2001. This meeting is said to be the beginning of a revolution in software development. Since then, the eXtreme Programming (XP) enthusiasts are setting the pace for development of state-of-the-art digital technologies. The above-mentioned agile methodologies are the most common ideas behind the XP approach.

The following declaration has been made at this meeting:

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

* Individuals and interactions over processes and tools,
* Working software over comprehensive documentation,
* Customer collaboration over contract negotiation,
* Responding to change over following a plan.

That is, while there is value in the items on the right, we value the items on the left more. [1]”

How does CI relate to the above declaration? Since collaboration with a customer is very important, the simplest way to collaborate with customers is to present the outcome of our work on a regular basis. The point is that, in order to show the customer a product, the code that is running inside this product has to work as the customer expects. The question arises, is it possible to deliver new functionalities, which were implemented in the code written a few hours ago, to the customer with a high degree of certainty in terms of its good quality. Of course, it is possible to achieve this. The only thing that needs to be done before handing a product off to a customer is to find issues in the code and fix them long before the customer will find these issues.

In the preceding subparagraph, is has been said that a product can be delivered to a customer as soon as an evidence on quality of the working code is provided. These are just some of the requirements for a software development process that allows to release new functionalities within the tight deadlines:

* The build process shall be done automatically following the strictly defined procedure,
* The test environment shall emulate the normal operating condition for a product as close as possible,
* Test scripts shall be written before the code is written and those tests should be automated,
* Integration and testing of the code shall happen several times a day,
* The simples solutions shall be implemented to meet today’s problems,
* Generation of business stories should be used to define the functionality,
* Shared code ownerships amongst the developers shall be promoted,
* An on-site customer for continual feedback and to define functional acceptance testing shall play a crucial role in the project. Those acceptance tests may be automated and built-in in the CI pipeline while being supervised by the customer.

With the above requirements, there are numerous iterations of software builds, each requiring testing. Once a developer starts writing every test case, it has to be ensured that it can be automated. Every time a change is introduced in the code, it shall be tested at the component level and then integrated with the existing code, which is then fully integration-tested using the full set of test cases. This gives continuous integration, by which it is meant that changes are incorporated continuously into the software build [2].

## Motivation to take Continuous Integration topic up

This paragraph is a quick summary of my engineering career path. I started my career here in AGH University of Science and Technology as a mechatronics engineering student, then in parallel to the studies, I have been an intern in Industrial Turbomachinery Systems department in Woodward Poland sp. z o.o. for 14 months, finally I ended up as an embedded software verification engineer in Aircraft Turbine Systems department in the same company. Oddly enough, I have never got a chance to work on a project that has been developed based on one of the agile methodologies. Due to the strict certification process, the projects I was involved in are mostly developed using the V-model software development life cycle. Each level in the V-model was divided into small activities that were realized using the customized waterfall models.

The interesting thing is that, while I was working on particular projects, I did not consider some project activities as problems that are mainly caused by some weaknesses of the V-model or waterfall model itself. Currently, I am aware that most of these problems can be easily addressed by introducing agile methodologies into the project activities. The breakthrough that has led to changes in my outlook on software development process was participation in the ISTQB[[5]](#footnote-5) Foundation Level course.   
I passed the certification and gained the practical knowledge of the fundamental concepts of software testing including people in roles such as testers, test analysts, test engineers, test consultants, test managers, user acceptance testers and software developers. The scope of ISTQB Foundation Level covers all software development practices including Waterfall, Agile, DevOps and Continuous Delivery [3].

There is huge potential for making use of agile methodologies, particularly the CI process, to enhance the process of embedded software development for aerospace applications. Thus, I have decided to develop and validate my own continuous integration tool. I did my best to deliver a solution that can solve many engineering difficulties I have encountered in the past. Furthermore, while I was working on the CI tool that is subject to this thesis, I was facing many challenges. Solving many of these problems was very demanding and time consuming. Thus, it should not be a surprise to anyone that, to complete a software project on time and on budget, there is a need to have a CI expert onboard, who is responsible for development and maintenance of the whole CI infrastructure dedicated to the project.

## Review of technological know-how for CI process

In the recent years, continuous integration has become a very popular cure to the common dysfunction of many software teams. Thus, it is obvious that the CI subject matter experts want to make use of their expertise for the commercial purposes. As   
a result, there is a lot of training courses teaching how to incorporate ideas of the CI process to an organization. These courses are often tailor-made and suit the needs of   
a particular organization. Participation is such courses itself can be very expensive.

On the other hand, there are some people who are still willing to share their knowledge for free. While I was working on this thesis, I found many useful information about the different aspects of continuous integration. There is a lot of articles on the software development process in the internet, which in my opinion are the most valuable source of information. The authors of these articles have been beginning from the same starting point as I have. Similarly, they had to develop   
a continuous integration tool from scratch. Most of the stories presented by them start with some simple CI solutions that got the management approval. Over the course of time, the more time was spent on the CI tools development, the more buy-in was received. Finally, these guys have led to fundamental changes in the entire process of software development. There are many examples that I am referring to, be the most flourishing one, which has significantly increased my interest in CI can be found at the following link: https://bulldogjob.pl [4].

In one of the preceding subparagraphs, the costs of implementation of agile methodologies in an organization was considered. It has to be kept in mind that these cost are certainly much higher. In many cases, software teams that are adopting agile are bound undergo a deep restructuring to start thinking in agile way. Based on the below examples of successful implementation of the CI practices in the world's largest high-tech companies, undoubtedly it is worth making financial investments to use the agile mindset on a daily based.

Here are 4 trailblazing companies that exemplify the possibilities of DevOps[[6]](#footnote-6) [5]:

* Amazon – on average, engineers are deploying code every 11.7 seconds.
* Netflix – known for its commitment to automation and open source.
* Facebook – known for its accelerated development lifecycle that meets consumers' expectations of software by bi-weekly app updates, effectively served notice and constant, rapid refreshes for mobile apps.
* IBM - It is estimated that 70-80% of teams are pure agile teams.

# Software development process

Software development process, also known as a software development life cycle, is the process of dividing software development work into phases. The phases contain manageable chunks of tasks that can be assigned to individuals responsible for a certain activity within the software development life cycle. Depending on available resources, type of a software project or a product and last but not least, software development model, different activities can be can be defined. The most common are:

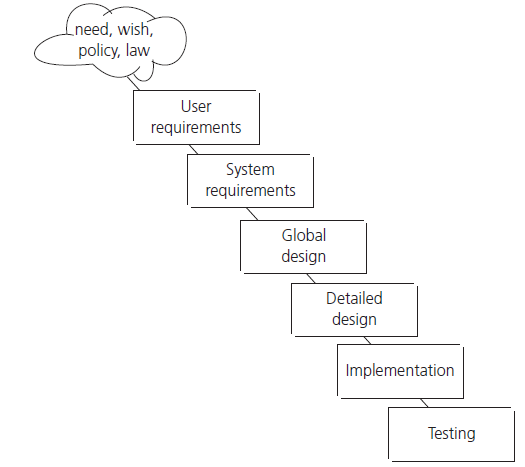
* Requirements definition,
* Software architecture design,
* Code development,
* Software build process definition,
* Testing,
* Debugging and bug fixing,
* Deployment,
* Configuration management,
* Maintenance.

In the next part of this paragraph, the above activities are going to be presented with an emphasis on their roles and time frames in software development models. Each of the activities may either have its own separate phase or be aggregated with another activities into one larger phase.

## Waterfall model

The waterfall model is one of the earliest models that have been defined. This model has a natural timeline and all task are executed in a sequence. At the top of the waterfall there is study of user requirements, then system requirement are defined. The waterfall flows down through the various project tasks. Once design is ready, development starts, which in turn flows into build. In the final step testing activities are carried out.

The apparent risk that arises in the waterfall approach is a likely event of finding bugs in the testing phase close to the end of the project life cycle. In general, with this model it is difficult to get feedback passed to any preceding phase in the waterfall. There are also additional difficulties if there is a need to carry out numerous iterations for a particular phase.



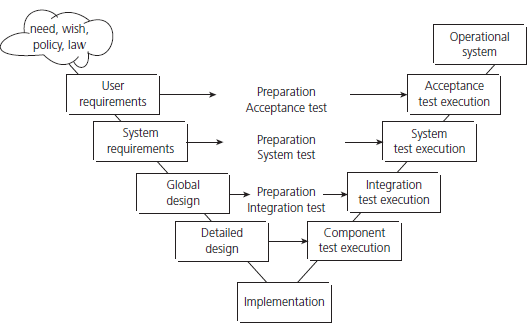
Fig, 1. Waterfall model [2].

Nonetheless, in some cases the pros of the waterfall model outweigh the cons. This model is efficient for small software projects that are carried out by few engineers. The other possibility of using the waterfall approach are projects that have well-defined requirement at the very beginning and these requirements are highly unlikely to be changed at any stage of the project.

## V - model

The V-model has been developed to address some of the problems that are being experienced using the waterfall approach. Bugs are found too late in the life cycle, as testing is not involved at the early stages of the project. The other problem associated with late testing is added lead time difficult to estimate. The fundamental principle provided by the V-model is to begin testing as early as possible in the life cycle. The model is also aimed at showing that testing is not only an execution-based activity and it defines a variety of testing activities that need be performed before the end of the code development phase. Ideally, these activities should be carried out in parallel with development activities.

The V-model illustrates how testing activities, which are validation[[7]](#footnote-7) and verification[[8]](#footnote-8), can be integrated into each phase of the life cycle. Each phase of the V-model has its own test level comprises of a group of testing activities that are organized and managed together. Validation testing takes place especially during the early stages (e.g. reviewing the user requirements), and late in the life cycle (e.g. during user acceptance testing). Verification tasks exist mostly in the middle stages of the V-model. However, in practice, a V-model may have more, fewer or different levels of development and testing.

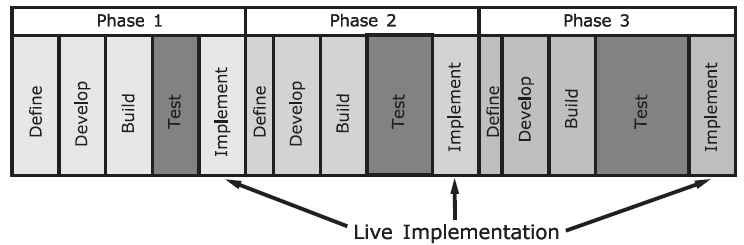


Fig, 2. V - model [2].

The V-model approach is a highly disciplined software development model. it promotes precise design, carefully development, and comprehensive documentation necessary to build stable software products. Hence, this approach is widely used in the applications that require high degree of reliability, which is standard in medical or aerospace industry.

## Iterative life cycles – agile methodologies

The main purpose of iterative or incremental life cycles is cycling through a number of smaller, effective and result-driven life cycles phases for the same project instead of one large development activity.



Fig, 3. Iterative development model [2].

A common feature of iterative approach is that the delivery is divided into increments or builds. The main advantage of this approach is that iterative development can give early market presence with critical functionality. As a result, the customer can provide the development team with feedback on the product. It the early versions are not satisfactory, the user or system requirements can be redefined with a little impact to the project time lime. Besides the business value and fitness-for-use of the product that are continuously improved in the subsequent deliveries to the customer, each increment adds a portion of functionality in the overall project requirements.

From the testing perspective, subsequent increments require testing for the new functionality, testing of the existing functionality and integration testing of new and existing code. Agile enthusiasts tend to say that working software is the primary measure of progress. Thus, regression testing plays a crucial role in all iterations after the first one. In some versions of the incremental approach, each phase follows a ‘mini V-model’ with its own design, coding and testing activities.

There are couple examples of incremental development models:

* Rapid Application Development (RAD),
* eXtreme Programming (XP) ,
* Agile methodologies.

For the characteristic of the above models with an emphasis on agile methodologies see section 1.2.

The iterative approach provides greater flexibility throughout the development process, because it allows the software to quickly respond to changes in market requirements or business environment.

**else** **if** (receivedMessage **instanceof** ClientMessage\_SensorInfo && sensor != **null**) {

System.***out***.println("[Compute engine Runnable " +sensor.getSensorID()+"] ClientMessage\_SensorInfo message from sensor: " + sensor.getSensorID() + " has been received.");

SensorImpl received\_sensor = ((ClientMessage\_SensorInfo) receivedMessage).getSensor();

**if** ((sensor.getCoordinates().equals(received\_sensor.getCoordinates())) &&

(sensor.getSoftwareImageID().equals(received\_sensor.getSoftwareImageID())) &&

(sensor.getSensorState().equals(received\_sensor.getSensorState())) &&

(sensor.getLocal\_watchdog\_scale\_factor() == (received\_sensor.getLocal\_watchdog\_scale\_factor()))){

**Bibliography:**

[1] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mallor, Ken Shwaber, Jeff Sutherland: The Agile Manifesto. Snowbird, February 2001

[2] RBCS - Rex Black Consulting Services, Inc.: Software Testing Book, 2003-2011

[3] ISTQB Foundation Level 2018 in a Nutshell.   
Available: https://www.istqb.org/certification-path-root/foundation-level-2018.html (visited August 9th, 2018)

[4] Exploring Jenkins Pipelines: a simple delivery flow. Available: https://bulldogjob.pl/articles/726-exploring-jenkins-pipelines-a-simple-delivery-flow

(visited August 10th, 2018)

[5] 10 companies killing it at DevOps. Available: https://techbeacon.com/10-companies-killing-it-devops (visited August 10th, 2018)

1. CPU – Central Processing Unit [↑](#footnote-ref-1)
2. IT – Information Technology [↑](#footnote-ref-2)
3. CI – Continuous Integration [↑](#footnote-ref-3)
4. CD – Continuous Delivery [↑](#footnote-ref-4)
5. ISTQB - International Software Testing Qualifications Board [↑](#footnote-ref-5)
6. DevOps - Development and Operations. Focuses on culture that aims at raising awareness of potential benefits for an organization, which are the result of continuous deployment (consists of continuous delivery, continuous integration, and continuous testing) with the lean management principles. [↑](#footnote-ref-6)
7. Validation - aims at giving an evidence that the requirements for a specific use or application have been fulfilled. [↑](#footnote-ref-7)
8. Verification - aims at giving an evidence that the specific requirements have been fulfilled. [↑](#footnote-ref-8)