Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Master of Science in Engineering - Dependable Systems 30.0 credits

EVALUATING CHALLENGES, BENEFITS, AND DEPENDABILITY OF VIRTUAL AND PHYSICAL TESTING OF EMBEDDED SYSTEMS SOFTWARE

Wahaab Alhasan
wan16001@student.mdh.se

Examiner: Masoud Daneshtalab
Mälardalen University, Västerås, Sweden

Supervisors: Eduard Paul Enoiu and Wasif Afzal
Mälardalen University, Västerås, Sweden

Company Supervisors: Per Erik Strandberg and Jonathan Thörn
Westermo Network Technologies AB, Västerås, Sweden

14/02/2022
Abstract

Software testing is a widely used quality assurance activity and often starts from the early development stages. However, starting early in the development process raises difficulties and challenges practitioners must deal with; most typically, hardware is not available in the required quantities, and there is the risk of damaging the hardware while testing. Emulating the physical hardware into virtual versions is a popular approach to overcome the mentioned obstacles. This master thesis, carried out at Westermo AB, investigates the differences between the physical and virtual hardware used for embedded system software testing and the possible benefits of combining both hardware versions in a hybrid system. Investigating differences between the hardware options helps identify which type of tests are more suitable in physical versus virtual hardware. The selected method for this thesis is a case study, starting with a pre-study phase investigating how other industries tackle the difficulties and challenges mentioned. Further, data were collected from two sources, historical test reports, and a questionnaire. The historical test reports showed a timing difference between virtual and physical hardware, but the link-up/link-down time is slower in physical hardware. The questionnaire also confirmed the timing differences as a significant challenge often experienced by engineers. Another challenge highlighted by the questionnaire answers is that "false positives" are typically caused by virtual hardware, where issues do not always turn up due to the virtual nature of communication. Another difference proven from this thesis is that virtual hardware is more failure-prone during the early stages of testing than physical hardware. The hybrid system could be advantageous in various ways, such as quantitatively expanding the current test systems and increasing test coverage. Future work could contribute with a proof-of-concept implementation of the hybrid system to confirm the advantages and demonstrate the third option of a test system.
Acknowledgment

Firstly, I would like to express my gratitude to all my supervisors during this thesis work. Per, Jonathan, Wasif, and Eduard, all of you have been such a great support during this work. Thank you for guiding me in the right direction. This could not have been accomplished without your supervision.

As this thesis concludes my studies at Mälardalen University, I would like to dedicate this degree to my parents and family, who have been the biggest supporters of my life along the way. Thank you for guiding me in the right direction!

I would also like to dedicate gratitude to Westermo for allowing me to do this work at your site. To all colleagues, you embraced me with knowledge, experience, and a great laugh when it was needed. Thank you all, from the bottom of my heart!
Contents

Acronyms 1

1 Introduction 2

2 Background 4
  2.1 Testing Embedded Systems 4
  2.2 Industrial Context 6
  2.3 Test Systems 9
  2.4 Virtualization of Network using GNS3 11
  2.5 Dependability 12
  2.6 XiL Testing Approach 15
  2.7 Digital Twins 17

3 Related Work 18

4 Problem Formulation 21
  4.1 Expected Outcome 21
  4.2 Limitations 21

5 Methodology 22
  5.1 Pre-study Phase 22
  5.2 Data Collection Phase 22
  5.3 Data Analysis Phase 23
  5.4 Reporting Phase 24

6 Ethical and Societal Considerations 25

7 Results 26
  7.1 Overview of Test Systems 26
  7.2 Analysis of the Fail Verdicts 28
  7.3 Failure In Time (FIT) Measurement 32
  7.4 Questionnaire Analysis 33

8 Discussion 36
  8.1 Findings Related to ReQ1 36
  8.2 Findings Related to ReQ2 37
  8.3 General Findings 38
  8.4 Validity Threats 39

9 Conclusions and Future Work 40
  9.1 Future Work 41

References 45

Appendix A Questionnaire 46

Appendix B Invalid Verdicts 47

Appendix C Test verdicts overview 49

Appendix D Test verdicts overview (2) 51
List of Figures

1. Overview of V&V activities .............................................. 4
2. GitLab feature driven development .................................. 6
3. Ring and Star topology .................................................. 7
4. Closed and Open state MRP ........................................... 8
5. Example of a physical test system ................................... 9
6. Example of a virtual test system .................................... 10
7. Dependability tree ....................................................... 12
8. Reliability measurement illustration ................................. 13
9. Dependability threats ................................................... 14
10. Context visualization of XiL approaches ......................... 16
11. Typical software reliability curve .................................. 20
12. Case study method illustration and its phases .................... 22
13. Fail verdict over the branch timeline ............................... 29
14. Example of a hybrid test system ..................................... 38
15. Invalid verdicts - physical test systems ......................... 47
16. Invalid verdicts - virtual test systems ............................. 48
17. Pie charts - virtual test systems .................................... 49
18. Pie charts - physical test systems ................................... 50
19. Bar charts - physical test systems .................................. 51
20. Bar charts - physical test systems .................................. 52
21. Bar charts - virtual test systems .................................... 53
22. Bar charts - virtual test systems .................................... 54

List of Tables

1. Summary of XiL approaches ............................................. 16
2. Archival data collection - artifacts ................................... 26
3. Invalid verdicts for each test system ................................. 32
Acronyms

**DUT** Device Under Test 9, 10, 26, 27, 29–32, 34, 36–38, 40, 46

**FIT** Failure in Time 3, 13, 24, 26, 29, 32, 36, 39, 40

**FRNT** Fast Reconfiguration of Network Topology 8

**GNS3** Graphical Network Simulator 3 11, 34, 37, 40

**HiL** Hardware in the Loop 15, 16, 18, 38, 39

**IOS** Internetwork Operating System 11

**IPv4** Internet Protocol version 4 11

**IPv6** Internet Protocol version 6 11

**ISTQB** International Software Testing Qualifications Board 15

**LLDP** Link Layer Discovery Protocol 8

**MiL** Model in the Loop 15, 16, 38

**MRC** Media Redundancy Client 7, 30

**MRM** Media Redundancy Manager 7, 30

**MRP** Media Redundancy Protocol 7, 8, 30

**MTBF** Mean Time Between Failure 12, 13

**MTTF** Mean Time To Failure 12, 13

**MTTR** Mean Time To Repair 12, 13

**NIC** Network Interface Card 10

**NIO** Network Input/Output 11

**QEMU** Quick Emulator 11, 37

**RSTP** Rapid Spanning Tree Protocol 8

**RT** Regression Testing 5

**SDLC** Software development life cycle 4, 15

**SiL** Software in the Loop 15, 16, 38, 39

**SRGM** Software Reliability Growth Model 19

**TRDB** Test Results Database 5, 13, 23, 26, 28

**V&V** Verification and Validation 4, 16–18

**VPN** Virtual Private Network 8

**WeOS** Westermo Operating System 6, 8, 9, 21, 28, 34, 35, 37, 40

**XiL** X-in-the-Loop 15, 16, 38
1 Introduction

Software plays a significant role in our society. It is a crucial ingredient in many devices we depend on in either work context or in our daily needs. Software is present, not only in the computer at our work desk, but even in the car, phone accessories, or the tiniest devices in our lives like the electric toothbrush. The working and behavior of such devices and many other complex ones, such as network routers, financial networks, the Web, airplanes, spaceships, etc., are defined by software [6]. The software can either result in simplicity and efficiency or cause unexpected frustration for the users [39].

Historically, bugs or errors in the software have been a leading reason for catastrophes. They cause millions of dollars in losses or worse, loss of life in some cases. A study conducted by Wong et al. [60] investigated some of the most catastrophic accidents where software was responsible. An example often mentioned in academia is Therac-25, where at least five patients died due to malfunctioning of the radiation therapy system between 1985 and 1987. Therac-25 was a software-controlled system that produced overdoses of radiation, and it was estimated to be 100 times more than the predetermined dose. A similar accident reappeared in March 2001 at Panama’s largest radiation therapy institution. The software was responsible for overdoses of radiation again. Five of the patients died immediately and nine over the next few years. After investigation, one of the leading issues were found that caused the overdose is the software program did not output any warnings regarding improper usage. A third example worth mentioning is the loss of Ariane 5, where the malfunctioning was traceable to a software failure. On June 4, 1996, Ariane 5 was to be launched into low Earth orbits. Thirty-seven seconds after the take-off at an altitude of about 3500 meters, the rocket deviated from the expected path, broke up, and exploded. The development costs were estimated to be 7 billion dollars. The errors that arose were traced to the active and backup computers on-board; both collapsed once a failure input occurred, resulting in a total shutdown of both computers and consequently loss of attitude control. Furthermore, in 2018, another software miscalculation caused the death of a pedestrian by one of Uber’s self-driving cars. The sensors recognized the pedestrian, but due to "false positives", the software assumed the pedestrian as an empty soda bottle on the road and kept driving [40, 60].

According to Wong et al. [60], these accidents mentioned above have resulted in lessons everyone involved in software development should be aware of. One example is that testing for the bad is as necessary as testing for the good, meaning that testers should test the software for unexpected and expected behaviors. If the software were tested for what it should not do, that might have helped avoid the overdosed radiation or the Ariane 5 incident. Another lesson mentioned by Wong et al. [60] is that testing virtually or in a simulated environment are not always accurate enough to launch a system. Software should be tested on physical equipment whenever possible to ensure proper behavior. The "false positive" is a typical result from performing only virtual testing. Success during simulation does not always mean success during real-life operations [60].

Software testing is by far the most used quality assurance activity. However, testing in the early development phases implies several difficulties for testers. For example, physical devices may not be finalized to execute tests, the number of physical devices may not be enough to perform sufficient testing, or physical devices may be too expensive to risk possible damage caused by failing tests [15]. Solving these difficulties often requires new testing methodologies and efforts. Typically, simulations or usage of virtual hardware can replace physical hardware until it is available.

This master’s thesis, conducted at Westermo Network Technologies AB, aims to investigate the differences between physical and virtual hardware dedicated for the software testing of embedded systems at Westermo. Further, what benefits can be provided if both hardware, physical and virtual, were combined into a hybrid test system?

Westermo specializes in designing and manufacturing data communications products used for high-demand critical systems such as electricity centrals, transportation industries, water systems, and more. Developed products are used in harsh environments such as critical temperatures, moisture, electrical interference, and vibrations. However, the parameters mentioned above require reliable, robust, and safe products to perform the expected functionalities [56]. As part of the testing phase,
virtual and physical hardware is used for software testing, and both hardware options stand out with their advantages and disadvantages.

The selected methodology for this thesis work is a case study, inspired by the guidelines of Runeson et al. [37]. A case study has been selected due to the exploratory nature of this method. The thesis starts with a pre-study phase, identifying the state of virtual and physical testing practice in other industries. The next phase is data collection, where two methods are applied: archival data collection and questionnaires. The archival data is used to compare how the testing behaved in virtual and physical test systems. Meanwhile, the questionnaire’s main target is to address a hybrid test system’s possible benefits. Data analysis is conducted based on the test results’ deviations, aiming to identify differences between the hardware approaches, and the responses from the questionnaire may illustrate how a hybrid test system can contribute to software testing.

Earlier research in virtual and physical testing from robotic, aviation, and automotive industries has discussed the testing approaches’ differences, advantages, and disadvantages. Papers often propose a transition from virtual testing to physical testing as virtual testing does not always reflect the actual behavior and is less reliable. Simulations tend to raise unrealistic behaviors that can be inadequate compared to the real world. Furthermore, a reality gap between virtual and physical testing is one of the main challenges experienced by practitioners, resulting in less trustworthiness and an exceeding gap between simulation and physical testing. Besides these challenges and disadvantages, simulated hardware provides numerous benefits for the testing phase. To mention a few: comprehensive test coverage, testing from early stages of development independent of the availability of physical hardware, improved debug sessions, usability at different levels of testing, and finally, simulated hardware can easily be controllable when needed in situations where, for example, a restart is desirable. However, much research has been done comparing virtual and physical testing, often in automotive, aviation, and robotics. Yet, less research is done where dedicated test systems, consisting of physical or virtual hardware, are used for embedded system testing. Further, a hybrid solution combining virtual and physical hardware as one entity has rarely been addressed in research.

This thesis emphasizes that virtual test systems are used more frequently than physical ones. Further, a reality gap considering timing issues, frequently occurred in several test executions. The timing issues were also confirmed as a significant challenge experienced by testers and developers at Westermo. Another finding highlighted in this work is regarding the reliability term. The Failure in Time (FIT) measurement emphasized that physical test systems are more vulnerable and failure-prone than virtual test systems. A failure of a test system is assumed to be a state where the executed test could either receive a verdict Pass or Fail. The test execution could not continue and had to be interrupted at this stage. Most failures discovered in the physical test systems are because physical hardware could not reset when needed. Regarding the hybrid test system, it could be advantageous allowing more test coverage than the typical test systems separately and extend the current systems to allow further emulation of customer-specific cases.

The remaining sections of this thesis are divided according to the following: background (Section 2) provides the essential knowledge to understand this thesis work, including the state of practice in virtual/physical testing. Related work (Section 3) discusses different approaches with their benefits and drawbacks. Problem formulation takes place in Section 4, including the expected outcome and the limitations in advance. The method is thoroughly described in Section 5. Ethical and Societal Considerations are explained in Section 6. All results of this thesis are presented in Section 7. Discussion, conclusions and future work are presented in Sections 8 and 9.
2 Background

The background section is divided into subsections, providing essential knowledge to understand this thesis work. First subsection 2.1 explains the fundamentals regarding embedded systems and testing of embedded systems. Subsections 2.2, 2.3 and 2.4 provide an understanding of the devices produced by Westermo and the test systems used for embedded system testing. In subsection 2.5 the dependability term are explained thoroughly, meanwhile remaining subsections such as 2.6 aims to familiarize the reader with the state of practice in virtual and physical testing.

2.1 Testing Embedded Systems

Verification and Validation (V&V) is a procedure widely used in software engineering to build quality into products life cycle [54]. V&V is a series of activities to be implemented during the Software development life cycle (SDLC) to ensure that each phase corresponds to the specified requirements and expectations. The term verification refers to evaluating the system design, architectural design and module design during the development cycle to prove that it meets the expectations and requirements during testing phases. Meanwhile, validation aims to test the final solution in an acceptance test, at the end of the development to ensure the functionality meets the requirements [2]. An illustration of the V&V model can be seen in Figure 1.

![Figure 1: An overview of V&V activities in software engineering](image)

V&V, which often is perceived as the V-model, complement the waterfall concept. In fact, both concepts have similarities such as being a top to bottom model. The main difference between the models is that V&V establishes a relationship between each development phase and corresponding testing phase [11]. As seen in Figure 1, software testing is a process and part of the V&V in SDLC. The formal definition of software testing, according to ANSI/IEEE 1059 is "testing is the process of analyzing a software item to detect the differences between existing and required condition (that is, bugs) and to evaluate the features of the software item" [1]. Software shall be reliable, therefore no unexpected or unintended behaviour is accepted. The testing process consists of executing the program to find errors, identify defects, and discover unintended bugs [43]. Testing is also normally divided into several stages, starting from unit testing and finishing at acceptance testing. Developers commonly perform unit testing to test each unit individually to ensure the right behaviour. To clarify a unit in a simplified way, the unit can be further explained as a specific function in software or a specific electronic component in hardware. Units individually may not produce unexpected behaviour, but when integrating few units can cause new results. Therefore, integration testing aims to compile several units at once. System testing is mostly focused on the entire system, aiming to conform the actual behaviour and performance with expected behaviour and derived system requirements. A final acceptance test is performed to ensure if the device is ready to release. At this stage, nonfunctional requirements may be tested [15].
Testing Embedded Software

In combination, hardware and software components create embedded systems, which are widely used in computing systems that continually and autonomously control and react to the environment. The market for embedded systems is steadily growing, and the need for embedded systems is increasing rapidly, leading to more sophisticated software modules to handle hardware components in demanding environments and strict constraints [41], such as limited memory, CPU usage and energy consumption. Embedded software differs from regular software in such a way that the majority of embedded software are "non-human interfaces" [19, 3]. Testing a non-human interface requires different test approaches, for example developing more software or using particular applications to deal with the non-human interface. Simulating electrical signals to use as input for embedded software is one method used to test embedded systems [3]. Another widely used method for embedded software testing is the usage of specialized compilers and development software, which enhance debugging of embedded software [41]. Many practitioners widely discover that a challenge within embedded systems software testing is observing concurrency faults. Yu et al. in [63] point out "the frequent use of interrupts for timing, sensing and I/O processing" can normally cause concurrency faults. Furthermore, concurrency faults can be divided into several types. To mention a few, data races, atomicity violations and deadlock are common [63]. A data race is described as when a single variable receives at least two accesses, and one of them is specified for writing. The race will occur when no synchronization between the accesses is deployed [12]. Atomicity violations are also a memory-related concurrency fault, often occurring when inconsistency between data access has occurred. To illustrate the fault more precisely, when a series of instructions is about to be executed, the developer may exclude other underlying instructions which use the same memory. In such cases, atomicity violations are exceeded [20]. Deadlock often occurs during the lack of synchronization between several programs using the same resource, leading to a rise of concurrency between the programs [64].

Such faults are not always predictable, and in several cases, they may be produced without any visible failures. Testers, in general, try to solve these occurrences by repeating the execution, aiming to reveal the fault during at least one execution. Such approaches may not always be sufficient to deal with these faults. A particular problem with this approach is described by Yu et al. in [63], faulty outputs are not guaranteed to occur even when concurrency faults exist within the software. Testers in such cases shall, according to Yu et al. "observe the simultaneous transmissions and not just rely on the presence of incorrect outputs" [63].

Testing Embedded Hardware

Embedded hardware testing differs from embedded software. From a broad perceptive, hardware testing can be divided into two test methods, Design verification and hardware testing. Design verification aims to confirm the design with the specification, meanwhile hardware test is where the designer induces faults to conform to the functionality [28, 41]. Using fault generation vectors, to inject faults into the circuit and simulate the behaviour is also a known method to identify the functionality and observe the output. For several circuits, test vectors can be used to monitor the output and compare to the original design [28]. However, embedded hardware testing is not in scope for this thesis.

Regression Testing (RT)

Software often needs new modifications due to a changing environment, and customers need new updates, concepts, or technologies. Adapting to these changes requires growth in the existing functions, interfaces, or components. Consequently, the software expands beyond the original state. Software regression means when new modifications are added to the current edition [25]. Regression testing (RT) is one of the most used retest techniques for software, which is applied after software modifications have been implemented. RT may be used during the development phase when new bugs or errors have been introduced in the tested program [26, 29], but is also very useful in a maintenance phase when new modifications, updates, or changes in user specification shall be implemented [29, 59]. Software modifications may involve creating new functions or files to improve a previous version or implementing new logic to correct a bug or error. These changes
may be performed in tested and approved software. Approved software has been tested according to a test plan, therefore using RT shall not affect previous test cases. RT is used to ensure that new modifications have not affected the software [26]. RT is widely used at Westermo, where one test framework executes test cases and orchestrates the hardware used for testing and Suitebuilder, another internal developed tool, prioritizes tests in test suites for RT to take place [44]. Test results are recorded in a Test Results Database (TRDB). The TRDB produces several reports each night to update the tester regarding the nightly session. Results are normally presented in six categories:

- Pass: tests pass when the software being tested are performing as expected.
- Fail: tests fail when the software being tested are not performing as expected.
- Invalid: tests were classified as invalid due to unhandled exception or failures in the test framework or test environment.
- Unmappable: lack of physical resources can cause an unmappable tests.
- Unloadable: errors in code or syntax can cause an unloadable tests.
- Skipped tests: tests that could not be executed due to time limitations or low-prioritized by the SuiteBuilder [45].

2.2 Industrial Context

This subsection familiarizes the reader with Westermo’s speciality in designing and manufacturing data communication devices.

Westermo specializes in designing and manufacturing robust data communication devices to be deployed in harsh environments [44], where consumer-graded equipment are not sufficient to manage such applications. Devices such as *Industrial Ethernet Switches*, *Routers* and *Modems* are aimed to be in industries such as transport, water and energy, as these services are part of mission-critical applications. The harsh environment in these sectors requires equipment that can withstand extreme temperature ranges, vibrations, dust and moisture.

Many devices developed by Westermo run the specified software implemented in-house, Westermo Operating System (WeOS). WeOS is developed in Git branches, where a master branch is the main branch, and all developers extract their branches dedicated for their tasks. This approach allows the master branch to stay as healthy as possible; meanwhile, developers create new features. Once a developer is finished with his development, a merge request is sent to integrate the feature back to the master branch. Figure 2 illustrates this approach, where a master branch is planned to be released. FeatureBranchA and FeatureBranchX are extracted by a developers to perform their tasks and later on, sent back by a merge request ¹.

![Figure 2: The GitLab feature driven development flow with an example of master branch and two development branches [45]](image)

Some of the products satisfies the IEC 61850-3, and IEEE 1613 standards for communication [56]. The standard IEEE 1613 covers the requirements of environmental and testing of communication

¹https://martinfowler.com/articles/branching-patterns.html
networking devices [7]. Meanwhile IEC 61850-3 is an international standard which aligns communication protocols for devices in power plant and substation environments ². Ethernet switches are used in wired networks to connect wired devices across a network. When data packages are sent between the devices, the embedded system of the switch recognizes the destination and delivers to the aimed device, that is possible because the addresses of each device is stored intelligently in the switch. Wired devices are preferable in an industrial environment due to their reliability and higher communication speed.

Switches have the restriction of not being able to exchange data outside the dedicated network. That restriction requires a device able to read IP addresses, which is a significant advantage for routers. Routers connect several networks and allow data packages to be sent between networks and, consequently, devices. The router identifies networks by their IP addresses and forwards the specified data packages to each network [53].

Such industries rely on these network devices to be faultless to avoid unexpected events. Network errors such as data package losses for more time than expected can cause expensive shutdown processes or stop in manufacturing lines. Devices used for networking shall be reliable and robust to achieve such applications [56].

Network Topologies

A network topology describes how network communication occurs between devices. The most common network topology is wired topology, but wireless topology is also used. Wired topology is further divided into several topologies such as Star topology and Ring topology. Star topology is when several devices are connected into a single central point (e.g. Ethernet switch) where all data passes through the central point before reaching the targeted device (see Figure 3b). Star topology offers advantageous robustness. If one device is disconnected for some reason, the network data can still reach out to other devices due to their connections. Single point of failure is also common in a star topology. All devices rely on the central point, and failures affecting the main point will affect the connected devices [42]. Ring topology is constructed in a closed-loop manner, where each device is connected to two other devices. Data packages are sent in a ring manner through all devices until it reaches the desired destination, see Figure 3a. In the majority of ring topologies, data packages are sent in one direction. Such an approach is called "unidirectional". "Bidirectional" is the second approach where data packages can be sent in both directions [42, 57]. Figure 3a represents the unidirectional approach.

![Figure 3: The most common networking topologies, Ring topology and Star topology](image-url)

[²https://webstore.iec.ch/publication/6010]
Media Redundancy Protocol (MRP)

Redundancy in the topologies mentioned above is one of the main objectives of reaching reliable communication between devices. Media Redundancy Protocol (MRP) is a redundancy protocol aiming to handle redundancy when part of the ring, connecting devices, is broken. According to MRP, the ring shall contain a Media Redundancy Manager (MRM) and at least one Media Redundancy Client (MRC). MRP have two states, Closed or Open. A closed state means no connections between the devices are broken. In such case, the MRM appoints the secondary port in "blocking" mode to avoid data packages being in a loop throughout the topology, see Figure 4a. The open state occurs when one of the connections between the MRCs is broken or if one of the nodes are powerless. In such case, the MRM appoints the secondary port in "forwarding" mode, see Figure 4b. For redundancy case, the MRM can appoint a secondary port to became primary. Such a case occur when the first appointed port fails operation [57].

![Closed state MRP](image)

(a) Closed state MRP

![Open state MRP](image)

(b) Open state MRP

Figure 4: Closed and Open state MRP

In order to verify the redundancy in MRP several types of tests can be executed. A typical test case is link breaking between devices. When link is broken, the reconfiguration time is of interest. The time aspect will vary depending on which device model is used and the type of link between devices. To verify the redundancy, a test case could be designed to break one link at the time between two devices [57].

Security is an essential attribute which is fundamental when sending private data over public network. A key concept to enable secure communication is using Virtual Private Network (VPN) tunnel. The VPN technology was mainly developed to ensure security with authentication and encryption which could deny access for confidential information [47]. The WeOS software is compatible to a open source OpenVPN suite in order to provide VPN tunnel services [57].

MRP is one of many examples of protocols used by Westermo to reach reliable communication between devices. Other examples such as Fast Reconfiguration of Network Topology (FRNT), Rapid Spanning Tree Protocol (RSTP) and Link Layer Discovery Protocol (LLDP) are also widely used in the field [57].

---

3https://openvpn.net/
2.3 Test Systems

In order to test the reliability in software, redundancy and the protocols mentioned above, test systems are developed and available to testers and developers to perform tests and modifications. Test systems are created in two versions, physical test systems and virtual test systems. Software and hardware can be tested individually to detect shortcomings, but integration testing which includes both software and hardware are even more important to detect combined-failures. The purpose of having test systems is to imitate customer installations, testing in different manners, such as manually or automated testing, and test customer specific cases. The amount of test systems are a restricted factor, for instance by efficiency and limited resources. Such boundaries allows only limited amount of test systems. Each system contains 4 to 25 hardware devices [46, 49].

The rest of this subsection aims to familiarize the reader with the construction of virtual and physical test systems.

Physical Test Systems

Physical test systems are constructed in a manner to emulate a network topology, where Device Under Test (DUT)s such as routers and/or switches are connected together by links. The links are cables between ports in the devices which could be DSL, Ethernet and serial cables.

![Figure 5: Example of a physical test system (Photo from [46] used with permission)](image)

Figure 5 demonstrates an example of a physical test system which uses nine DUTs. Each DUT runs the associated Westermo software WeOS. Custom-built nodes could also be part of the physical test systems. Such nodes can be manipulated to simulate specified behaviours which may be encountered in the field. A typical node is link breaker, which are able to demonstrate cable wear-outs, rust or other damages caused by the human factor [46].

Virtual Test Systems

The development and testing of embedded software are often dependent on the availability of the target hardware. The hardware preparation involves designing, simulation, testing components and architectures, circuit schematics, implementation, and product finalization. In addition to the stages mentioned, hardware development may also be affected by external factors such as component deficiencies or failures during testing, which causes unexpected harm to the product. In order to adapt to such adversities, system emulation and virtual environments allow software developers and testers to perform intended progression without access to the physical hardware. Another substantial advantage that can be extracted from the emulator is the safe and secure environment to perform unproved development and testing without harming specific hardware. Emulators in today’s market are well suitable for different versions of hardware and operating systems. Therefore software development can be tested for various criteria such as scalability,
reliability, and functionality. Such advantages expand the usage of virtualized environments and offer feasible testing platforms for embedded system testing [38].

Sampath and Rao in [38] clarifies the terms virtualization and emulation as "abstraction of one or more computer resources to achieve the behaviour of the desired system" respectively "emulation refers to the replication of functions of a system by another, so that the emulator acts similar to the emulated system." [38].

In the context of Westermo, a typical emulated device is a switch. The software emulation of the physical switch is the skeleton of the virtual switch. The next integral part of a virtual switch is the Network Interface Card (NIC), where the link between the NIC and virtual switch is a software emulated link. The host processing capabilities are the only factor to restrict the bandwidth of such a link. The emulated link can be adjusted individually for each NIC, where an upper limit, for instance, could entail balanced traffic within the virtual switch [55].

Figure 6 demonstrates an example of a virtual test system. In this example, four emulated DUTs (D1-D4) forms a ring with three connections between each DUT. Link breakers (X) are part of the ring to demonstrate a lost connection between the DUTs. D1-D3 and D2-D4 are connected cross-like with associated link breakers. D5 and D6 are outside the ring. Meanwhile, a PC is connected to all devices.
2.4 Virtualization of Network using GNS3

A Virtual Network is a logical software-based entity containing the infrastructure and functionality of a simulated network. The term Simulation can be described as an imitation technique where actual or imaginary objects can be simulated by a computer to create a system and observe its behavior. When aggregating the described definitions above, a simulated network can be defined as an imitated entity, where network objects such as routers, switches and hosts are interacting [14].

Network simulators are the common platforms to create a network topology and emulate the behaviors of the network. Simulators are widely used in academia for teaching purposes, designing, and quality assurance activities. Various simulators are available on the market, such as Graphical Network Simulator 3 (GNS3), Network Simulator version2 (NS2), Network Simulator version3 (NS3), M5 Simulator, and so on. Simulators can be categorized based on their simple or complex features, functionalities, open-source or commercial. For instance, a complex simulator may emulate several types of network devices and communication protocols. Meanwhile, a simpler simulator can be used for minor network topologies. For the usage of this thesis work, GNS3 is the typical simulator at Westermo. Besides being a powerful simulator, GNS3 also provides a graphical interface that can visualize the emulated network devices and the operation. Based on the survey conducted by Dayanand et al. in [14], GNS3 offers features such as high complexity and quality of network designs, emulation of multiple vendor’s devices, compatibility with physical networks and devices, and packets can be captured and analyzed with an inherent aid, Wire shark 4.

GNS3 is a python based open-source simulator that allows emulation of network simulations. In order to emulate routers or other networking devices, an Internetwork Operating System (IOS) image have to be supplied by users. Such images can be retrieved from Cisco or other vendors [14]. In the Westermo context, IOS images are available for Westermo products. Additionally to the IOS images, GNS3 are also compatible with virtual machines such as VirtualBox and Quick Emulator (QEMU). The virtual connectivity between the devices, in GNS3, are according to protocols such as Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6). Beyond the virtual devices, GNS3 also can link virtual interfaces to physical interfaces in a PC. Such an approach allows connecting a virtual network to physical hardware, a router, switch, or a second PC. The configurable device which allows such a solution is Cloud Node. The cloud node is not specific hardware, rather software that provides wide options of Network Input/Output (NIO) connections. Cloud nodes can be configured to communicate with physical hardware via the PC’s Ethernet adapter [32].

4https://www.wireshark.org/
2.5 Dependability

Dependability is a broad term, often mentioned as an umbrella term, which describes the ability of a system to deliver the intended service to its users. Dependability can be extracted into three fundamental branches, as emphasized in Figure 7. The Attributes describes the fundamental properties of a system. Based on the system or application, one or more attributes may be included to evaluate the system’s dependability. Dependability Means expresses the techniques used during the development of a dependable system to ensure dependability. Threats are the chain that highlights when something went wrong within a system [8, 16].

![Dependability Tree](image)

Figure 7: Dependability tree showing the branches of the main term and its elements [8]

The attribute of interest for this thesis work is reliability which described in the subsection below. Further, a brief introduction to the dependability threats are presented.

Reliability

Reliability describes the continuity of correct services within a system. This term emphasizes system operation without failures during a given interval of time [16]. The mission-critical applications require reliable devices during operation. Westermo develops devices expected to operate in a harsh environment without interruptions. Therefore test systems must ensure the ability to perform reliable testing. Similar to availability, reliability is a measurable term. Dubrova, in [16], says that reliability expresses the probability of correct services and can be measured by the number of failures during a given time interval, where the system is expected to function correctly [16].

Software reliability has its origin from hardware-related reliability models. The most common failures in hardware are often related to physical wear in different shapes, for example, corrosion, shock, or effects of temperatures, unlike software, where most faults are related to design or implementation. One simple measure which are often applied for both software and hardware is the Mean Time Between Failure (MTBF) with the corresponding acronyms Mean Time To Failure (MTTF) and Mean Time To Repair (MTTR), see Equation 1 [33].

\[
MTBF = MTTF + MTTR
\]

However, according to Iannino and Musa in [30], time, in software reliability, can be illustrated in two ways when discussing failures over a time interval, either mean-value function or failure intensity function. Mean-value function illustrates the average failures related to each time point, meanwhile failure intensity function represents the change of the mean value function, alternatively
the number of failures per unit time. Iannino and Musa further express the failure intensity as a derivative of the mean-value function with respect to time [30].

Figure 8: This figure emphasizes the relation between the acronyms mentioned in Equation 1

Figure 8 shows the relation between Equation 1 over a time interval. Time is not specified in this case, but normally it can be measured as days or even years, depending on the system under operation. The system in this figure operates normally until a failure appears, therefore the acronym MTTF. Once the system is not operating and down for repair, MTTR is the measurement of interest, which emphasizes how long it takes to get the system back in operation mode. The third acronym MTBF is the measurement between each failure.

It is established that MTBF measurement are uppermost related to reliability, but availability measurement can be further extracted from reliability. Pressman and Maxim in [33] says the following regarding software availability: "Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as":

\[
\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \times 100\% 
\]

However, beyond the equations 1 and 2, Pressman and Maxim suggests an alternative measure of reliability, which is Failure In Time (FIT). FIT is a statistical measure that emphasizes how many failures appear in a component over a specified operation time interval. Pressman and Maxim give an example where the operation time is 1 billion hours, and one FIT is equivalent to one failure in every billion hours of operation [33].

The FIT measurement is the relevant one for this thesis. The measurement can be applied to test systems to measure the failures over a nightly testing session, one specific test system (physical or virtual), or over an entire branch timeline. This measure can further hint about the most failure-prone test system compared to the remaining systems.

### Dependability threats

Dependability threats can be considered as a chain of multiple events causing system failure. More precisely and according to Avižienis et al. "... failure is an event that occurs when the delivered service deviates from correct service" [9]. Furthermore, failure within a system is a progressive transition, from delivering correct service to incorrect service. Service outage is the state where incorrect services are delivered, and service restoration is the state when the system is restored to deliver the correct service. When the service fails, at least one system’s external state is deviating from the correct service state. This deviation is called an error, and the cause behind the error is called a fault. Errors may not always cause failure within a system. Sometimes, the error does not reach the system’s external state, and therefore no failure may be caused. For the case of faults, they are active when they do cause an error. Otherwise, a fault can be dormant [9].

The definition of threats terminologies is further visualized in Figure 9. In the case of this thesis, the threats chain can be used in two sequences. One for the software development and the failures
regarded. The second one is more focused on the test systems themselves. Test systems fails to deliver correct service when they assign a test execution the verdict *Invalid*. According to [45], tests can receive such verdict due to unhandled exception or failures in the test framework or test environment. We assume this to be a failure state within the test systems. All failures, from both sequences, are reported in the TRDB.

Figure 9: Illustration of the dependability threats chain
2.6 XiL Testing Approach

The remaining subsections aim to familiarize the reader of this thesis work with the state of virtual and physical testing practice, addressing different test approaches well-established in other industries. Advantages, disadvantages, area of use and context visualization is also discussed.

As earlier mentioned, testing is an activity to establish a relationship between all development phases [11] and ensure quality of developed products [54]. But testing during the development phases may introduce challenges to deal with, such as physical devices not being finalized to execute tests, the number of physical devices may not be enough to perform sufficient testing, or physical devices may be too expensive to risk possible damage caused by failed testing [15]. X-in-the-Loop (XiL) is widely used in different industries and for several purposes with derived approaches such as Model in the Loop (MiL), Hardware in the Loop (HiL), Software in the Loop (SiL). The "in-the-loop" term indicates the relationship between the embedded system itself and the applicable environment [23].

Model-in-the-Loop (MiL)

As described in earlier sections, an embedded system is a composition of software and hardware used for specified environments. In most cases, a model of environment and model of the physical hardware is enough to verify the functionality of the modeled system [52]. MiL is commonly used during development phases since no software code or finalized hardware is necessary to apply MiL testing. MiL is a derived testing approach from Model-based development [23], therefore it is hard to mention MiL without mentioning Model-based development Model-based development is widely used in several industries and applications, allowing engineers to graphically model and simulate the behaviour of systems to evaluate certain functionalities, as well as an understanding of the model in the early design phase to enhance further development [52]. However, Model-based development is not in the scope of this thesis work.

Hardware-in-the-Loop (HiL)

The basic definition of the term HiL has been provided by Ledin in [24]: "Hardware-in-the-Loop (HiL) simulation is a technique for performing system-level testing of embedded systems in a comprehensive, cost-effective, and repeatable manner". Embedded systems are not always easy to test in their operational environment due to various reasons; therefore, a simulation of the environment can significantly improve testing [24]. During the early stages of the development, MiL testing may be efficient enough to perform tests at a high level. But as the systems mature, the need of using physical hardware for the test is inevitable. HiL provides the ultimate combination of both physical device and virtual testing, allowing developers to emulate a system by using a physical device and the remaining subsystems virtually within a closed-loop simulation. Several industries rely on HiL simulation to achieve progressiveness in testing from early stages of development. To point out few industries, automotive, aerospace, and marine. HiL contributes with even more advantages such as cost-effectiveness since less physical hardware is used. The comprehensiveness of HiL is also very advantageous, allowing testers to simulate a broader range of operating conditions in each domain. From a safety point of view, HiL simulation can be used in learning processes for human operators in industries such as aerospace (e.g. flight simulators for pilots) or automotive domain (e.g. car crashes) [18]. HK Fathy et al. provides an example of HiL simulation for a car suspension system where a physical microcontroller was used as an active suspension controller to estimate and command the states of the virtual suspension. The model was intended to estimate vehicle inertia, suspension stiffness, suspension damping and road characteristics. Such implementation gains knowledge and provides background for further calibration and optimization to be deployed before installation in actual vehicles. The HiL simulation demonstrates an advantage of low-cost testing before accurate vehicle testing is performed [18].

Software-in-the-Loop (SiL)

A widely used XiL testing approach in industries such as automotive is SiL. The broadly and commonly used definition of the approach is that software implementations are not applied to
specific hardware. Instead, it is running on a PC in a simulated hardware device [15, 50, 10]. Such an approach provides an advantage where the lack of available hardware devices cannot hinder the software’s ongoing development and testing. Furthermore, SiL can be further used for software verification during the SDLC [22]. Several test results cannot be fully reliable when conducted in the SiL approach, such as performance and reliability tests, as these tests are dependent on the use of hardware. According to International Software Testing Qualifications Board (ISTQB) in [4], the most recommended testing in SiL is functional, interface and regression testing [4].

To summarize the XiL approaches sufficiently and for the context sake, Table 1 visualizes each approach and the corresponding requisites to achieve a sufficient level of testing.

<table>
<thead>
<tr>
<th>XiL</th>
<th>Hardware</th>
<th>Software</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIL</td>
<td>Virtual</td>
<td>No</td>
<td>Virtual</td>
</tr>
<tr>
<td>HIL</td>
<td>Physical</td>
<td>Yes</td>
<td>Virtual</td>
</tr>
<tr>
<td>SIL</td>
<td>Virtual</td>
<td>Yes</td>
<td>Virtual</td>
</tr>
</tbody>
</table>

Table 1: Summary of XiL approaches

Meanwhile Figure 10 represents each XiL approach and the corresponding V&V phase. As often mentioned, testing shall be integrated within the development and from an early phase. Such an approach allows integrating testing into a system, component, hardware, software and models.

Testing from an early phase provides advantages during the development of products. Failures in the behaviour of a system or unexpected interactions between components can be discovered, repaired and evaluated from an early concept phase to avoid time loss and production costs [35].
2.7 Digital Twins

A well-established approach to perform simulations is *Digital twin*, which allows testers to replicate physical systems digitally. Such a system can further contain static and dynamic properties of the physical entity. Digital twin replication can be assumed as real-time systems; any changes or updates in the physical system shall be automatically implemented in the digital twin replica.

A digital twin is asset-specific storage and provision of models, which can be used for several purposes by developers, testers and operators. The approach provides the possibility of performing simulations parallel to operations to conform assumptions, and such an approach is needed to strengthen the V&V activities to reach more reliable systems and effective testing. According to the survey conducted by Löcklin et al. in [27], the Digital twin approach has been widely used in V&V in three areas. Exploratory investigations, testing and formal proving. The exploratory investigations can be summarized as an approach to understanding how a system behaves in certain states. Such an approach can be investigated thoroughly by data evaluations of the system in experimental procedures and by observing the system behaviour. Formal approving with digital twin is mostly done mathematically to prove a system behaves in a certain way. The most practical approach for this thesis is testing with a digital twin, which practitioners broadly use. Digital twin in testing can be used to replace missing devices, unmade devices or devices scheduled for maintenance or can not be used for other purposes. Digital twin is a replacement for such cases [27].

The twin concept was firstly exercised by NASA’s Apollo space program, where they built two identical space vehicles. One of these vehicles operated in space as expected. At the same time, the second vehicle acted as a mirror for the first one. This twin replica was placed on earth and was used for simulations and predicting the conditions of the other one on mission. The second example of the digital twin, mentioned by Boschert and Rosen in [13] is the *Iron Bird*. The Iron Bird is a ground-based facility including engineering tools and test equipment for prototyping and integrating different aircraft systems and new designs. The Iron Bird, similarly to NASA’s invention, acted as a verification mirror to confirm new functionalities and characteristics of systems, components and discover incompatibilities [13].
3 Related Work

This section presents the related work in virtual and physical testing, highlighting the benefits of different approaches.

Roles of Virtual and Physical Testing

Wilkinson in [58] discusses the roles of physical testing and virtual testing, emphasizing how the virtual domain has expanded, which allows developers and testers to perform various complex tasks and processes by simulation. The virtual domain expansion could therefore extend the range of attributes that simulations can offer. However, simulations can be costly, time-consuming, and less reliable than physical testing in some attributes. The type of testing shall be discussed from an early development phase when designing the test and gathering the expected outcomes. Based on what attributes to test, each testing approach differs, which is also indicated by [48].

For instance, testing fatigue in an exhaust system is challenging to be performed virtually due to material properties and crack propagation at different temperature ranges. In addition, human responses and characteristics of system parts are difficult to simulate in a virtual environment and in the worst cases, they could cause unreliable test results and therefore decrease the confidence in virtual testing. The most suited approach to deal with these difficulties is physical testing. Additionally, physical testing can be used for performance purposes and the response of complex systems. Wilkinson indicates that virtual testing may be performed for the understanding of system behaviour, interactions and sensitivity [58]. To achieve sufficient product development, it is essential to understand what each testing approach can offer in terms of benefits and, even more importantly, how both methods can be combined [58, 48]. Integrating both approaches is highly rated, and a hybrid solution can be beneficial in many aspects, such as shortened testing time and confidence in testing [58]. Tahera et al. in [48] suggest an overlap between the two test approaches, aiming to achieve a combined testing approach containing both virtual test and physical tests, but also pointed out that virtual testing may be validated to ensure reliable results and to gain more confidence in testing results [48].

In [21], Himmler emphasise a transition from virtual testing to HiL testing can contribute to reduced costs during the testing stages. Producing physical hardware dedicated for testing are both expensive and inflexible when several hardware are needed for multiple projects. By emulating the hardware into virtual versions, such challenges can be overcome and testing from an early stage of the V&V-cycle are achievable. Since physical hardware is a limited resource, HiL testing should be performed on regular basis even 24/7 if possible. In the aviation industry, virtual hardware used for testing and HiL testing is commonly used and closely interlinked. A typical case is testing starts with virtual hardware and slowly being replaced by physical hardware. Hybrid system which includes multiple virtual hardware and physical hardware is also mentioned by Himmler as a regular solution [21]. In [31], the authors used the HiL approach and found features such as test automation, stimulus generation and fault injection to be easily addressed for both software and hardware.

Simulation-based Testing in Robotics

Simulation-based testing is a popular approach among practitioners in the robotic field. The papers [5, 36, 51] demonstrates the vital role of simulation and virtual testing due to the flexibility and safer testing approach rather than physical testing. In [36], Robert et al. conducted an industrial case study investigating the feasibility and effectiveness of virtual tests in a virtual environment. The subject of the study is an agriculture robot, used in farming for autonomously weeding. The researchers aimed to conclude the benefits and drawbacks of simulation-based testing, by executing eighty different tests, and each test ran five times, resulting in 400 runs of test cases. Simulation-based testing demonstrated that most software issues revealed during physical testing could also be found in virtual testing. Simulations also revealed unexpected faults, behaviours and difficulties that could be produced during the operational missions, such as U-turn failure, which is critical for the agriculture robot. The specified failure could be obtained during simulation when multiple tests are performed but may be harder to reveal during field tests as it requires a broader range of
tests over a more extended period of time [36].

Timperley et al. in [51] investigated the popularity of using simulation-based testing. In fact, 85% of the participants used simulation during testing for different purposes such as component testing and multi-robot testing. The study also demonstrated that simulation testing is preferable when physical or real environment testing is not available. One of the discussed challenges within simulation testing is the reality gap. Several practitioners experience an exceeded gap between simulation and physical testing. Simulations may cause unrealistic behaviour, which can be inadequate in comparison to the real world [51].

Simulated Hardware Used for Embedded Software Testing

Embedded software tested in simulated hardware has been discussed by Engblom et al. in [17]. Simulated hardware provides benefits such as comprehensive test coverage, improved debugging sessions, and usability at different levels of testing. Simulated hardware can easily be controlled when a restart is desirable or restricted access to the physical hardware. During the integration testing, testers can use simulated hardware to run multiple versions of the operating system. System setup time during system-level testing can efficiently be reduced with simulated hardware. Testers can use scripts to configure the hardware instead of having physical cables between devices. The scripts can further be modified for new conditions or saved for future usage. Emulating the simulated hardware into multiple versions increases the amount of test coverage, and different tests can be executed on various hardware simultaneously. Fault injection in simulated hardware is desirable due to the non-destructive output. Physical hardware could be exposed for permanent damages during testing, either unintentionally or intentionally. Fault injection could also be made in loops to repeat the faults continuously during a broader range of time, and testers can record the behaviour of the system to observe the outputs during different time intervals. The economic aspect of simulated hardware is also a beneficial argument. Simulated hardware allows several developers to work continuously from different workbenches. The costs of having specified inventory or maintenance activities can be reduced, and software and hardware can be developed independent [17].

Software Reliability

Software development is often divided into four phases: specification, design, coding, and testing. Software faults and errors occur mostly during the first three phases, which later becomes a failure and appears during the testing phase. Therefore, software verification is one of the most important phases to ensure software quality. The most known characteristics of software quality is functionality, usability, maintainability, probability, efficiency and reliability. Reliability is a term that is often taken for granted when discussing software quality. According to Yamada in [62], software reliability can be measured if the total number of errors can be estimated with high accuracy during the testing phase. Software Reliability Growth Model (SRGM)s are available to assess the reliability during the testing phase. SRGMs establishes a relationship between the detected error or failure occurrences and the time interval of the testing.

In [61], Wood measures the defect detection in a test environment. Therefore, he gives three options to measure time: execution (CPU) time, calendar time and number of tests run. The calendar time estimates how many defects can be found during a specified time interval, for example, if the machine was running for two days or 48 hours. Meanwhile, execution time is how much CPU time was used. However, the number of tests run in a great measure if all tests had a similar probability of detecting defects, which is unrealistic in some cases. Some test suites can execute 100 tests in one hour, while other more sophisticated may require 24 hours to execute [61].

In [34], Quyoun et al. visualize the software reliability in curves according to Figure 11. Since the software does not experience damages from the outside, wear-outs, or ages (like hardware), its functionality will not change over time until intentional changes/upgrades are made. Therefore, the failure rate for software is often high during the early stages of development but decreases towards the useful lifetime (idealized curve in Figure 11). However, if new changes/upgrades are made, the failure rate is compromised until a new state is declared (actual curve in Figure 11).
Summary

We found papers in this section in the robotic, aviation, and automotive industries, but it was harder to find research close to our subject. However, there seems to be a sufficient amount of papers discussing the main benefits of virtual versus physical testing. To our context, a lot of recommendations and benefits can be generalized. These are the benefits and challenges we can learn from:

Papers propose that virtual testing can be used for interaction and understanding systems rather than performance testing. The primary outcome that can be generalized is that testing on simulated hardware can be time-consuming and less reliable than the physical hardware. Therefore, it should be discussed from an early development stage what to test virtually or physically to ensure reliable and successful testing. Simulations are also highlighted as a safer testing approach, not causing any destructiveness to the hardware. Further, testers must consider the reality gap between virtual and physical testing. Practitioners often experience an exceeding gap, resulting in unrealistic behavior within simulations.

Moreover, simulated hardware provides the benefits of comprehensive test coverage and easier access to the hardware. Simulated hardware further offers relief when it comes to test system setups. Scripts can be used to configure the hardware and links between devices instead of having cables. Physical hardware is often implemented with one specific operating system. Meanwhile, virtual hardware can run multiple versions. Simulated hardware can be emulated into multiple versions to increase the test coverage, allowing several testers to perform their duties instead of waiting until physical hardware is available. Another advantage worth mentioning is the fault injection which can be safely implemented without compromising the health of the hardware.

We highlighted that most software faults occur in the early stages of development regarding software reliability. If these faults are not addressed early, they will propagate into failures during the testing phase. In order to measure the reliability, Wood in [61] suggests three options to measure the time. Calendar time and the number of test executions are the options that mostly fit our context since we attempt to measure the failure rate of the test systems.
4 Problem Formulation

This thesis investigates the differences between virtual and physical hardware used for embedded system software testing and possible benefits from a hybrid system consisting of both virtual and physical hardware.

Physical hardware is the closest to actual customer experience, but physical hardware is restricted by many factors such as quantity and cost. A test developer may implement tests requiring fifty devices, such physical hardware is difficult to procure. However, virtual hardware is not restricted to quantity, and all developers can easily access such test systems. Furthermore, virtual hardware can be flexible, allowing different test scenarios and a safer test approach. If one physical hardware is available, and the tester emulates the remaining hardware virtually, what type of test results can such an approach provide, and most importantly, are these results reliable enough?

Historical test results from nightly testing will be investigated to unravel the differences between each hardware solution to investigate these inquiries deeply. The analysis can provide an insight into what benefits and limitations each hardware solution offers. Further, the total failures of each hardware approach will be measured to address how many failures occur in each test system over a software branch timeline.

The following research questions aim to address these inquiries:

- **ReQ 1:** What are the main differences and what challenges are experienced when virtual hardware and physical hardware are used for testing embedded systems software?
- **ReQ 2:** What benefits can be provided from a hybrid test system consisting of virtual and physical hardware?

4.1 Expected Outcome

This master thesis focuses on comparing and evaluating test results from the nightly testing of virtual hardware and physical hardware, to point out the differences and benefits of each approach to set up hardware. Once a comparison is achieved, a further investigation will be conducted to address if a hybrid system containing both virtual and physical hardware can be beneficial.

4.2 Limitations

Few limitations are considered in this thesis work. First is the evaluation of test verdicts. Only one WeOS branch is selected to be investigated, even though comparing more branches may contribute to more detailed and versatile results. Due to time limitations, only one branch has to be considered. Each branch developed over a more extended period may contain more than 10,000 test verdicts. Therefore, comparing several branches is an infeasible task during this thesis work.

Next limitation regards the questionnaire, which will be conducted as part of the data collection. The questionnaire will be submitted to employees at Westermo, the number of answers we receive may be less than expected.
5 Methodology

The selected methodology for this thesis work is case study, inspired by the authors Runeson et al. and the guidelines they report in [37]. The case study is the suitable method because of the exploratory purpose of this thesis. The term exploratory is defined by Runeson et al. in [37] as “finding out what is happening, seeking new insights and generating ideas and hypotheses for new research”. This thesis aims to investigate the differences between two testing approaches and challenges experienced (ReQ 1) and if a new test approach is beneficial (ReQ 2) in the context of Westermo. The four phases included in this thesis are visualized in Figure 12.

Figure 12: Case study method illustration and its phases

5.1 Pre-study Phase

The pre-study phase establishes the state of practice in virtual and physical testing, investigating the most popular approaches, benefits, drawbacks, and recommendations from research papers. This phase is reported as part of the Background Sections 2.6, 2.7 and the Related work Section 3.

Papers were found from several databases and search engines, the most used ones is: Google, Google scholar, IEEE Xplore, Malardalen University Library, SAE Mobilus, SpringerLink, DiVA and ScienceDirect.

The most used keywords were Virtual testing, Simulation based testing, Physical testing software engineering, Hardware-in-the-Loop, Hardware emulation, Embedded system virtual testing and Virtual testing software.

5.2 Data Collection Phase

Two methods are used during the data collection phase. The first is archival data collection and the second is a questionnaire. According to Runeson et al. [37] the archival data is typically a third-degree type of data, meaning that it is often vital to support this method with other data sources. Therefore, a questionnaire is selected to support possible findings from the archival data.
Archival Data Collection - Procedure

The archival data’s primary target is to address the differences between the virtual and physical test systems.

Archival data are collected from Westermo’s test framework, which they use daily to support their testing activities. An official test results data base (TRDB) is developed by the company to store all test results from every nightly session. The TRDB further produces multiple test reports every night to be reviewed by the involved individuals. However, the TRDB is developed according to the GitLab feature-driven development, meaning that a branch of the database can easily be extracted and modified without affecting the master branch. Therefore, a local TRDB was extracted to allow access to all archival testing data from the company of interest, including software development branches and testing reports.

After the configuration of the local TRDB, one software development branch was selected for investigation, named FeatureBranchA. This feature branch was decided for multiple reasons: (1) The branch was not under development anymore, meaning that any experimental implementations would not cause harm; (2) the branch was continuously tested throughout the entire development time; (3) both virtual and physical hardware were used to test the selected branch.

After selecting the development branch, the focus shifted towards the test systems used to test the branch. Every test system was investigated individually and quantitatively first. Statistics were collected and presented in two different chart forms as seen in Appendices C and D. The Pie charts in Appendix C demonstrate a comparison between each test system and how many verdicts were assigned during the entire branch timeline. Meanwhile, the bar charts in Appendix D give a detailed demonstration of how many verdicts were assigned every night a test system was part of the testing.

Questionnaire - Procedure

The second data collection method is a questionnaire, aiming to support the findings from the archival data collection and gather opinions and experience from the employees at Westermo regarding challenges experienced and benefits of a third option test system. The questions are formulated semi-structured to allow both open and closed responses. The form is attached in Appendix A.

The questionnaire consists of two sections, where the first one is mandatory to answer, and the second section is optional. The second section targets employees who have implemented a local hybrid test system on their work desks. Section 2 allows the employees to describe how they implemented the test system, what challenges they faced, and how the test results differed from the physical/virtual test systems.

The questionnaire is sent by the Microsoft Teams platform, one of the options for communication inside the company. Other available communication options are email or handing out a paper format. These options are not used.

Eleven employees received the form, and they had four working days to answer the questions. The questionnaire’s completion time is estimated to take 45-60 minutes.

5.3 Data Analysis Phase

The data analysis phase establishes the results of this thesis work. Therefore, the analyses are conducted in several stages:

Firstly, a brief overview of the selected development branch and all test systems involved during the testing. The overview gives a statistical illustration of each test system during the branch timeline. The overview are presented in subsection 7.1, Appendices C and D.

A Fail rate over the branch timeline is estimated from Appendix D and visualized in Figure 13. All intervals of the branch timeline are investigated qualitatively. The investigation aims to seek root causes behind the Fail verdicts and if the root causes differed in virtual versus physical test
systems. Specific test cases are selected for investigation. Justification for these cases are presented in subsection 7.2.

The next stage of data analysis is the execution of the FIT measurement. The FIT measurements aim to statistically show how many times a test system fails to operate as expected. Expected operation is when the test system executes tests and assigns Pass or Fail verdicts. When Invalid is assigned, the test environment or framework fails to handle the test.

The total number of Invalid verdicts are firstly discussed in subsection 7.1, shown in Appendix B, C, D and summarized in Table 3. The failure ratios of each test system are estimated by applying the formula below:

\[
\text{Failure ratio} = \frac{\text{Total number of Invalid verdicts}}{\text{Total number of test executions}}
\]  

(3)

The FIT measurements are presented in subsection 7.3.

The last stage of data analysis is questionnaire interpretation. The questionnaire is shown in Appendix A meanwhile summary and interpretation of the results are in subsection 7.4.

5.4 Reporting Phase

Reporting is the last phase of this thesis work, aiming to summarize the results from the data analysis phase and discusses recommendations, future work, and the thesis’s outcome. They are presented in Sections 7, 8 and 9.
6 Ethical and Societal Considerations

The ethical aspect regarded in this thesis is straightly related to the questionnaire conducted as part of the data collection phase. It is mentioned a few times that employees in Westermo are the subject of the questionnaire. For that matter, the first two questions will not be presented or cited during results or any other section of this thesis. However, it is imperative to keep the answers anonymous during the discussion. Furthermore, the questionnaire results are anonymized to ensure answers are not traced to a specific individual.

Beyond the questionnaire, a non-disclosure agreement (NDA) was signed before starting this thesis work. The NDA ensures that all sensitive data regarding Westermo will not be explicitly highlighted. Therefore, all data collected will be generalized as much as possible.
7 Results

The following section presents the results of this thesis work in four subsections. Firstly a presentation of the selected branch and each test system used for the testing. In subsection 7.2, the collected archival data are analyzed. The FIT measurement are presented in subsection 7.3. Analysis of the questionnaire is presented in subsection 7.4.

ReQ1: What are the main differences and what challenges are experienced when virtual hardware and physical hardware are used for testing embedded systems software?

The first research question is divided into two inquires: 1) what are the main differences between the virtual and physical hardware which are used for testing embedded systems, and, 2) what challenges are experienced when virtual and physical hardware is used for testing embedded systems. First inquiry are answered in subsections 7.1, 7.2 and 7.3.

ReQ2: What benefits can be provided from a hybrid test system consisting of virtual and physical hardware

7.1 Overview of Test Systems

The following subsection, with the Appendices B, C and D provides an overview of the test systems which were used for the selected branch testing. The quantitative data in the Appendices illustrates the test verdicts in each test system. The Appendices compare all considered verdicts: Pass, Fail, and Invalid.

Presentation of the Selected Branch

The selected branch to investigate is named featureBranchA. The branch was tested continuously for 99 days. Further, it has been run on multiple virtual and physical test systems and for a broad date interval. The test systems execute specified test cases nightly, and all test results are stored in the official TRDB of the company. Majority of the test executions have fallen under the categories Unmappable, Unloadable, and Skipped tests. These verdicts are not considered during the analysis since they do not reflect either the behavior of test systems or the software under test. The verdicts to be investigated are Fail and Invalid. Invalid verdicts are assigned when an unhandled exception or failure occurs in the test framework or environment. The Invalid verdict will be the foundation for executing the reliability measurement FIT.

Table 2 presents each test system, the number of DUTs in every test system, the days each test system began and finished its part of the branch testing, and the number of tests executed by each system.

<table>
<thead>
<tr>
<th>Test systems</th>
<th>DUTs</th>
<th>Test date</th>
<th>Test executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical1</td>
<td>4</td>
<td>62 - 98</td>
<td>3436</td>
</tr>
<tr>
<td>Physical2</td>
<td>8</td>
<td>54 - 74</td>
<td>2311</td>
</tr>
<tr>
<td>Physical3</td>
<td>18</td>
<td>83 - 98</td>
<td>1439</td>
</tr>
<tr>
<td>Physical4</td>
<td>16</td>
<td>16 - 76</td>
<td>7245</td>
</tr>
<tr>
<td>Virtual1</td>
<td>6</td>
<td>50 - 98</td>
<td>7593</td>
</tr>
<tr>
<td>Virtual2</td>
<td>6</td>
<td>34 - 98</td>
<td>7074</td>
</tr>
<tr>
<td>Virtual3</td>
<td>6</td>
<td>0 - 36</td>
<td>3770</td>
</tr>
<tr>
<td>Virtual4</td>
<td>8</td>
<td>26 - 79</td>
<td>5675</td>
</tr>
</tbody>
</table>

Table 2: Summary of test systems artifacts, whether it is physical or virtual, number of DUTs used, test dates, and the number of test executions for each test system
Quantitative Overview of Physical Test Systems

Physical1 is one of the physical test systems which uses a total of four physical DUTs. Tests for Physical1 were executed between days 62 and 98 for the selected branch featureBranchA.

Appendix D represents the test results from this test system, and Appendix C shows a comparison between the Pass, Fail, and Invalid test verdicts. These verdicts were deployed to a total of 3436 test cases. Majority of the results went Pass, 154 tests received the verdict Fail, and 73 had the verdict Invalid. Fail and Invalid tests are most common during the middle of the branch as shown in Appendix D. Since the branch was under continuous development, Fail verdicts were common in that sense. Towards the end of the branch timeline, there is a decrease in Fail verdicts, and Invalid verdicts are negligible. It is also noticeable that the number of test executions decreased towards the end of Physical1.

Physical2 is the second physical test system which contains eight physical DUTs. Tests were executed at a shorter calendar interval, between days 54 and 74. 2311 test cases were assigned the verdicts Pass, Fail, and Invalid. Pass test verdicts, when compared to Fail and Invalid verdicts, are significantly high as shown in Figure 18. The Pass verdict has been dominant throughout the entire testing when Physical2 has been used for the selected branch. Fail test verdicts are significant during the start of the testing but decrease relatively towards the end (see Figure 19).

Physical3 is the biggest physical test system, which uses 18 physical DUTs in total. Having this significant number of physical DUTs is beneficial for specific test cases, for example, when testing a customer-specific case or a redundancy protocol. The number of tests that received the verdicts Pass, Fail, and Invalid was totally 1439, the lowest number compared to all other physical and virtual test systems. Pass test results are dominant when compared to Fail and Invalid verdicts. The majority of all Fail and Invalid verdicts occurred mainly during the start of the testing and slowly decreased towards the end, see Figure 18.

Physical4 is the test system that was mostly used compared to all other physical test systems, with a total of 7245 test executions and 58 days. Physical4 provided inconsistent test verdicts as shown in Figure 18, where the number of Pass test executions still dominated. Meanwhile, Invalid verdicts are also high. The verdicts Fail and Invalid mainly occurred during the first half of the test period and slowly decreased towards the end of the testing, see Figure 20.

Quantitative Overview of Virtual Test Systems

Virtual1 is an entirely virtual test system that includes six DUTs. The number of Fail and Invalid verdicts are notably minimal (35, respectively 6) in comparison to the Pass test verdicts, which are 7552 (see Figure 17). Fail verdicts are widely spread on the entire time interval but converges towards 0 at the end of the testing on Virtual1, see Figure 21.

Virtual2 is the second virtual test system with the most dedicated test executions and was running for the longest time compared to the other test systems. According to Figure 21, Invalid and Fail verdicts mainly occurred during the beginning of testing and slowly decreased towards the end of the branch testing.

According to Figure 17, Virtual3 has provided the most non-Pass verdicts compared to the remaining virtual test systems. Fail verdicts frequently occurred during the mid of the branch. Meanwhile, the Invalid verdicts decreased towards the end of the branch (see Figure 22). The Fail verdict was still common across the testing period, and during some specific days, Fail verdicts exceeded the Pass verdict (from day 10 to 15), see Figure 22.

Non-pass verdicts in Virtual4 mainly occurred during the start of testing and, later on, converged towards non-existing. The number of Pass test verdicts is dominant when Virtual4 has been used, compared to the Fail and Invalid verdicts, according to Figure 17.
7.2 Analysis of the Fail Verdicts

This section analyzes the Fail verdicts to answer the first inquiry of research question 1. We investigate the differences between virtual and physical test systems when assigning Fail verdicts to test executions.

Test Cases to be Compared

The archival data used for data collection provided essential information regarding each test system and the tests they executed. As observed in Table 2 each test system executed thousands of tests within the selected branch. It was observed that several test executions provided more varying results than others. Therefore, the focus was shifted towards these executions and their verdicts. These executions are listed in the itemization below:

- F1: Undesired access to the local network and devices
- F2: Loss of logging messages
- F3: Aggregating multiple ports into one port to achieve redundancy
- F4: Extracting multiple switches from one switch
- F5: Redundancy protocols
  - F5.1: Fast reconfiguration when a link between devices is broken
  - F5.2: Reconfigure a local area network topology into a logical tree topology
  - F5.3: Redundancy between several routers if one router is unavailable towards a host. The host assumes all routers to be only one

The itemization above represents the test executions that are considered during the analysis. These executions were extracted from test reports acquired from the TRDB.

In order to appropriately address how each execution behaved during testing with a virtual or physical test system, it is crucial to understand what each test means. F1 is a feature in WeOS implemented and used to protect against undesired access to a local network or device. That is usually tested with packages sent around devices, and an undesired guest tries to sniff these packages.

Execution F2 considers a logging activity conducted between devices and hosts. Devices typically produce information regarding the device’s health, other components within the network, or general events. These log messages are saved in log files. The interest of investigating this execution is to compare the behavior of the logging activity in virtual systems versus physical systems.

F3 is a feature in WeOS, where the aim is to combine multiple ports with acting as one single port. This feature provides redundancy between ports if they fail to operate as expected. If one of the ports fails during operation, the available port primarily handles the data transmission. However, F3 can be used for further objectives, such as increasing the data transmission. The compelling aspect of this execution is to monitor the behavior of the ports. We are further interested in inspecting if these ports behave differently in virtual and physical systems.

Execution F4 is extracting multiple logical switches from a single switch.

The executions listed above can either be combined during testing or tested explicitly. However, redundancy protocols (F5) are executions that can be gathered in one group as several protocols are tested frequently with the same ambition, which is achieving redundancy differently. However, it is essential to distinguish between the subgroups F5.1-F5.3 as they ultimately require different setups and configurations. More on redundancy protocols are described in the background section.
Analysis of Selected Test Cases

In order to present how each hardware approached the software testing, a quantitative comparison between the executions are shown in Figures 17-22, and now Figure 13.

Figure 13: The behaviour of the "Fail" verdict over the branch timeline

Figure 13 represents the selected branch featureBranchA timeline in the X-axis and the average of Fail verdicts that occurred in the Y-axis. The X-axis is divided into five calendar days intervals. The selected branch was under continuous development and tested on 99 calendar days. The calculation procedure to estimate the average of Fail verdicts is according to below:

- Number of Fail verdicts occurred each nightly session in every test system
- The number of Fail verdicts expressed in percentage
- All failure percentages were then added together and divided on the number of test systems used for that specific night session

This procedure is similar to the FIT measurement, as described in subsection 2.5. In this case, FIT measures the Fail rate of each nightly session (calendar time) on test systems.

An instant difference observable from Figure 13 is that the Fail verdict varies during specific calendar periods and is less varying during later periods of the branch timeline. Further, software testing on virtual test systems started before the physical. The third observable difference is that physical test systems tend to have more peaks than virtual test systems, from day 50 and towards the end of the branch, even though both options of test systems are still below the 10% line.

Taking a closer look at the first interval between days 0-15, the test execution F3 caused most Fail verdicts. One hundred fifty verdicts appeared during these days concerning the test execution F3. When studying the test reports from these executions, it was observable that the ports inside the DUTs, to be aggregated caused the majority of these verdicts and was ongoing for multiple days in a row. When investigating possible root causes, the ports configuration was the main fault during software implementation, which propagated into failure during testing and caused Fail verdicts.

Between the interval of 16-25, physical test systems participated in testing the selected branch. When investigating the test reports, the combination of F3 and F5.1 caused most Fail verdicts during this period and was equally failure-prone in both hardware systems. Before a test execution starts, all DUTs are usually reset to prevent side effects from earlier tests. The reset process often
took more time in virtual test systems than physical (30 seconds difference in some cases). After the reset, the test scripts force all associated ports to link up.

A difference was highlighted here in physical devices; the ports responded within three minutes. Meanwhile, ports respond within less than one and a half minutes in virtual devices, even in these executions where the same number of ports were expected to link up. When investigating more test reports not associated with specific test execution, this behavior of reset and link-up/link-down times regressed showed up now and then.

An interesting takeaway note from this observation is that reset processes are faster in physical devices, but ports’ link-up/link-down time is slower in physical devices.

F1 is an execution that often requires data packages sent between devices and the host. This execution has performed better in virtual test systems than physical ones during this specified calendar interval, meaning fewer Fail verdicts occurred in virtual systems. These executions were often more time-consuming in virtual test systems than physical ones, even when similar data packages were sent between the DUTs.

When comparing execution F2, no differences were found between the test systems. F2 was executed more on virtual test systems and therefore had a higher Fail rate than physical systems during this calendar interval. Such execution is not dependent on a hardware feature, and timing-related issues did not occur particularly here. Similar tests, which are not dependent on hardware features, should be tested more frequently in virtual systems to allow more test coverage in physical test systems.

At the following interval between 25 and 35, both virtual and physical test systems had identical peaks but differed in the lower peaks. Test executions F3 and F5 had almost no failures during this period. In virtual test systems, only 7, and in physical, only 3. Arguably, the selected branch featureBranchA started to gain more stability.

However, the execution F4 behaved differently. A total of 44 executions received the verdict Fail in the virtual system and 13 Invalid verdicts, of 103 total executions. Meanwhile, only 17 Failed and Invalid were assigned to 48 executions in physical test systems. The difference here is that similar test executions did affect the physical test systems differently than virtual test systems. A dependency failure struck the physical test systems, which affected the remaining tests. The dependency failure started in one of the DUTs in test system Physical4. The associated DUT failed to reset and was unable to perform the remaining tests as expected. The dependency failure is shown in Figure 15.

Dependency failures have not been observable earlier, nor in virtual test systems. That is a noteworthy observation worth mentioning as one of the takeaways.

In the following interval, 35-45, there is a gap between days 36-38 in physical test systems. Tests from the selected branch were not executed these nights on physical hardware. However, test execution F1 never received a Fail verdict at any virtual test systems but had an Invalid run at physical test systems. A new dependency failure is discovered in the physical systems here. The dependency failure was traced to software implementation faults in the test environment, where integral commands could not execute during the automated testing. The verdicts were assigned five days uninterruptedly.

Moreover, the peak in days 41-42 (see Figure 13) was primarily due to execution F5 in physical test systems. This execution had a total of 60 Fail verdicts during these days. After investigating the test reports, most of the issues were due to failures in the physical DUTs.

In some cases, DUTs did not power up. In other cases, the test scripts failed to assign roles to each port or DUT. In Section 2.2, we discussed the configuration of MRP for instance. MRP requires one MRM and at least one MRC. These assignments frequently failed in the investigated calendar interval. Similar failures did not occur in virtual test systems.

Another takeaway from these findings is that DUTs not powering up when expected can occur, but rarely in virtual systems. In Section 2.3 it was discussed that devices are linked with cables.
Such cables are vulnerable and can strike several DUTs unintentionally. Similar vulnerabilities are often non-existent in virtual systems, according to Engblom et al. [17].

We compress the remaining testing days, 45 and towards the end of the branch, and describe the observations at once: Figure 13 indicates that physical hardware provides more failures in testing compared to virtual. For example, execution F1 did never fail in virtual during the remaining time of the branch but had few failures in physical test systems. Furthermore, we could not trace these failures to the hardware nature of the physical systems.

Test execution F2 had issues in both virtual and physical test systems. It was noticeable that some differences in the link-up/link-down of ports/DUTs, as they took more time in physical systems than virtual. The executions themselves also took more time (from that, the executions start until the verdict is assigned) in physical systems than virtual.

Execution F3 failed once in virtual systems and had more failures in physical systems. In virtual systems, the failure cause is loss of response from one of the DUTs. Similar issues did not occur in physical systems at this time interval but were observed earlier during testing. The failures in physical were instead because of errors within the software implementation.

The remaining executions F4, F5.1, F5.2, and F5.3 had close to no failures at all the remaining time of the branch.

To summarize the takeaways from this analysis concerning the research questions:

- An observed difference between the virtual and physical hardware is that reset of DUTs process is faster in physical hardware, but ports’ link-up/link-down time is slower in physical devices.

- The execution F1 always had better Pass-run in virtual systems rather than physical systems. However, the execution was often more time-consuming (e.g., taking more time from execution until a verdict is assigned) than physical systems.

- A difference mention-worthy is that DUTs loses power have occurred more in physical systems.
7.3 Failure In Time (FIT) Measurement

In this section, the FIT measurements are used to identify how many failures emerged in the test systems. As earlier described (in Section 2.5), the FIT measurement is a statistical measure that emphasizes how many failures appear in a device over the operation time.

In the context of this thesis work, a failure in a test system appears when the system cannot deliver the expected functionality. The expected functionality is assigning Pass or Fail verdicts to a test. When a test system fails to do so, Invalid verdicts are assigned to the execution. Therefore, we measure how many such verdicts (Invalid) occurred in each test system every nightly session.

However, the operation time can be estimated in various ways according to [61] such as CPU time, calendar time and number of test cases. Therefore, the selected operation time is calendar time (Nightly sessions) and the number of test executions to the time scale.

Table 3 illustrates the total number of Invalid verdicts, the calendar days each test system was under operation for testing, the total number of tests each system executed, and, in percentage format, how many of these verdicts were Invalid. It is also illustrated how many Invalid verdicts arose during the nightly sessions in Figures 15 and 16.

<table>
<thead>
<tr>
<th>Test systems</th>
<th>Invalid verdicts</th>
<th>Calendar days</th>
<th>Total test executions</th>
<th>Invalid/Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical1</td>
<td>73</td>
<td>36</td>
<td>3436</td>
<td>2%</td>
</tr>
<tr>
<td>Physical2</td>
<td>19</td>
<td>20</td>
<td>2411</td>
<td>1%</td>
</tr>
<tr>
<td>Physical3</td>
<td>26</td>
<td>15</td>
<td>1439</td>
<td>2%</td>
</tr>
<tr>
<td>Physical4</td>
<td>1589</td>
<td>58</td>
<td>7245</td>
<td>22%</td>
</tr>
<tr>
<td>Virtual1</td>
<td>6</td>
<td>48</td>
<td>7593</td>
<td>0%</td>
</tr>
<tr>
<td>Virtual2</td>
<td>143</td>
<td>64</td>
<td>7074</td>
<td>2%</td>
</tr>
<tr>
<td>Virtual3</td>
<td>488</td>
<td>37</td>
<td>3770</td>
<td>13%</td>
</tr>
<tr>
<td>Virtual4</td>
<td>318</td>
<td>53</td>
<td>5675</td>
<td>6%</td>
</tr>
</tbody>
</table>

Table 3: This table presents how many Invalid verdicts were assigned for each test system, for the selected branch.

The calculations which were made for Column 5 in Table 3 are according to Formula 3.

From Table 3, it is observable that two of all test systems caused more failures than others (Physical4 and Virtual3). When considering the figures 15 and 16 (Appendix B), it is further observable that these failures occur at the early and middle stages of the software testing.

The potential reasons behind these verdicts vary. In physical test systems, it was observable from the analysis in Section 7.2 that sometimes one Invalid verdict could cause a chain of multiple verdicts coming in a row. One failure could occur in the test system and cause the remaining failures. For example, on day 25 in Physical4, the test system had significant issues related to DUTs. One of the DUTs had reset issues and could not be used for the testing anymore. Similar issues reappeared on days 29, 31, and 35, where more than 200 verdicts were assigned repetitively. The DUTs had to undergo maintenance to resolve these issues.

In virtual test systems, the reasons behind test system failures differed compared to physical test systems. Issues, where DUTs could not undergo a reset did not frequently occur as in physical test systems. In virtual systems, the Invalid verdicts were often assigned due to failed link-up of ports, failures in test implementation, and failures when assigning the proper address to the right DUT or port.

The takeaway from this section is that physical test systems are more vulnerable to failures in physical parts of the system. Failures could be DUTs not able to undergo a reset and carries leftovers from earlier tests. Engblom et al. in [17] pointed out this aspect as one of the benefits of simulated hardware. A reset or restart of simulated hardware is easily controllable when needed. Furthermore, maintenance activities can be reduced, both time-consuming and economically beneficial. This conclusion has also been drawn by Engblom et al.
7.4 Questionnaire Analysis

Eleven employees received the questionnaire, and eight of them answered. That is a response rate of 70%.

The questionnaire provided varying answers for all questions. One of the respondents preferred using physical test systems, one preferred the virtual systems, two used their own desktop setups more frequently, and the remaining respondents (4 of 8) did not specify which test system option they used more.

A shortage of physical devices has been emphasized as one of the main challenges experienced by the employees, causing several difficulties during the daily work and testing activities. Employees often need to spend more time searching for devices rather than performing work duties. Generally, work gets halted, and sometimes, the work has to wait. One of the respondents answered: "Time will be wasted on searching of devices instead of the real work".

The shortage of devices has also caused other difficulties during the testing phase. For instance, to test customer-related cases or redundancy protocols, testers may depend on using more devices than in normal circumstances. The most significant number of devices that have been used for testing such inquiries is up to 50-59 by employees. Many physical devices are rarely accessible, which causes complications, delays, and shortcuts in testing accuracy. In addition to the customer-related cases, specific hardware features are also difficult to test without access to physical hardware. For instance, one of the responses has pointed out a bypass relay as one challenge. Tests related to this relay cannot be performed until the hardware is available.

However, physical test systems are more restrictive than virtual test systems. Test engineers often need to deal with issues related to the physical test systems, such as undocumented changes in connections between devices. One of the respondents expressed that sometimes new links are added or removed without updating the documentation regarding the changes. Similar to the test servers, new upgrades may be installed without ensuring the information has been reached to all personnel who use the test system daily. Physical test systems can be unreliable when someone has made some experimental connections without restoring to a known state before the nightly tests are started. Another respondent has also added inflexibility as one of the challenges experienced within physical test systems. The setup of these test systems is often fixed and cannot be changed to explore different scenarios. It is also time-consuming to configure a more extensive topology in a physical environment. Physical devices typically require more attention and maintenance due to the destructiveness which may emerge. The devices themselves can be tampered with, broken, or need a cooling environment as they generate heat. Fuses involved in testing can be destroyed, and the relays may cause high contact resistance.

Beneficial-wise, physical hardware resolves issues which often encounter in virtual test systems. One respondent highlighted the following: "Timing related issues and hardware incompatibilities can be found in physical tests that can be masked by virtual tests".

Physical test systems tend to ensure realism and reliable test results compared to virtual test systems. The reality gap is discussed in [51] as one of the challenges often encountered by practitioners in the robotic field during simulation testing. Virtual systems can cause similar behavior, resulting in a reality gap between the two approaches. One respondent said the following: "False positives in virtual systems can occur where an issue is not spotted due to the virtual nature of the communication".

The "False Positive" term is familiar from Savoca’s [40] perception of the Uber’s self-driving car accident from 2018. The self-driving accidentally caused a death of a pedestrian due to miscalculation of the software. The sensors in the car recognized the pedestrian, but due to "False Positive", the software assumed the pedestrian as an empty soda bottle on the road and kept driving [40].

Moreover, physical test systems are the nearest mirror of the actual products, allowing developers and testers to approach customer expectations closely. It also allows testing hardware-specific cases and leash hardware incompatibilities, which is eventually one of the main challenges within virtual test systems. A possible solution to overcome such adversity could be a hybrid test system, which
Wahaab Alhasan Virtual and Physical Testing

combines both virtual and physical devices entitled as one entity. The benefits of such a solution are testing hardware-specific cases and producing more extensive topologies to ensure better test coverage and more entities to test on. Additionally, the reality gap may be neglectable.

Moving over to virtual test systems in more detail, most respondents highlighted some obstacles when using the virtual test systems. For instance, running virtual testing require Python 2, and the latest Ubuntu versions do not support such old Python. This version of Python have reached it life end, and no more updates are available. Also, older versions of WeOS are not supported in virtual test systems, which reduces testing possibilities. Similar to physical test systems, the reality gap is also considered here. For example, the link breakers, which are typically devices in physical test systems, are only a software function in virtual systems, resulting in an unrealistic behavior. Timing issues can also occur in boot time, link-up and link-down time.

Beyond the complications mentioned above, one respondent emphasized that virtual test systems are generally more dynamic for building customer-specific topologies and do run redundancy protocols well. Virtual systems also tend to be more beneficial for build-verification/confidence testing, where new or changed functionality is implemented. Build-verification testing is an approach where a set of tests are executed on every new build to verify that the new build are functional and ready to be released for further testing. Also, virtual hardware is not limited in quantity, they cost nothing, and there is no delivery time. These advantages allow topologies to be widely extended and more test coverage. Virtual hardware is also easily to access for each employee, allowing developers and testers to work independently when needed. Similar conclusion has also been discovered by Engblom et al. in [17].

Furthermore, tests can be started, interrupted and debugged from a well-known and reproducible state each time. Citing one of the respondents: "Topologies can be \"infinite\" in number of devices, there are real customer applications with \{very many\} network devices that may be tested in an virtual environment with less effort than to do it with an physical environment".

A hybrid test system may contribute with a new dimension not fully explored yet with the separated test systems. We have already mentioned one advantage, which is allowing testing hardware-specific cases, but furthermore, the hybrid system could be used to extend the existing test systems. We identified in Table 2 how many DUTs each test system had. When considering the physical test systems, these could be extended further with virtual DUTs. For example, instead of 18 DUTs in Physical3, we could have up to 50 DUTs, and even more, with the remaining as virtual. This extension could be beneficial to increase the test coverage.

One employee highlighted the benefit of testing different versions of WeOS: "... with a hybrid system we can test both for \{older and newer versions\} of WeOS. This can also be done at the same time since we sometimes get cases that have both in a system."

The virtual test systems, in their present condition, can only run the new versions of WeOS. Meanwhile, physical test systems can run multiple versions of WeOS. As emphasized from the answer above, a hybrid test system could be beneficial to run different software versions in one entity. That is a typical custom case that should be considered during development.

When discussing the reliability of test results from the hybrid test system, all respondents agreed that it could not be worse than pure virtual test systems or better than purely physical. The hybrid system will stand in between the current approaches. Furthermore, the hybrid system could be used as a reference platform, beneficial for test compatibility between the existing approaches today. However, the reliability should be evaluated early to avoid uncertainty. Some tests must be performed many times on all three systems and compare the outcomes. This approach may also hint at what tests or areas of functionality are more appropriate for the hybrid system.

When we asked about the implementation of the hybrid system, we concluded that three employees did use an external platform, GNS3, to manage the virtual part of the test system and the internal tools to manage the physical part. However, the connection between the platforms was configured with Cloud node interfaces. More on that are described in subsection 2.4. Employees also emphasized the difficulties they faced during implementation and the uncertainty of introducing new error sources they were not entirely familiar with. Moreover, one respondent used the implementation
to perform exploratory testing of small topologies, while the second respondent tested redundancy protocols on slightly bigger topologies. The third employee only implemented a hybrid test system for proof-of-concept sake. None of the employees indicated any automated testing using GNS3.

To summarize the takeaways from this analysis concerning the research questions:

- One of the biggest challenges emphasized by most respondents is the shortage of physical devices. Typically it causes difficulties during daily work, delays, and shortcuts in testing accuracy.

- Customer-related cases and more extensive redundancy protocols are cases that often require large test systems. The shortage of devices results in shortcuts associated with these testings.

- Physical test systems are often vulnerable to undocumented changes, links added or removed, updated servers, and devices can be tampered with or broken. Furthermore, maintenance activities are required now and then.

- The setup of physical test systems is often fixed and cannot be changed to explore different scenarios.

- Physical hardware ensures correct behavior, no timing-related issues, and no "False Positives." Furthermore, it allows testing for hardware-specific features.

- Virtual systems do not support older versions of WeOS, always risk timing issues in boot time and link-up/link-down of ports.

- Virtual systems are beneficial for build-verification and confidence testing. Also, virtual systems offer more dynamics for building customer-specific cases.

- A hybrid system may explore a new dimension of test systems. It could allow both testing hardware-specific features, build-verification testing, and customer-related cases simultaneously.

- A hybrid system may also decrease the timing-related issues and enclose the reality gap.

- The hybrid system could also be used to extend the test coverage by allowing testing of both older and newer versions of WeOS.
8 Discussion

This section discusses the performed work from different angles by stating the study’s significant findings and the relevant results related to similar work. It also discusses the validity of the work, limitations and selected methods.

The main objectives of this thesis can be divided into three inquires. We looked for the differences between virtual and physical hardware used for embedded system software testing. The challenges experienced when using these hardware approaches (ReQ1) and the benefits of combining both hardware in one entity, a hybrid test system (ReQ2).

We investigated one software branch named featureBranchA because both virtual and physical test systems were used to test the implementations of this branch. Furthermore, seven test cases were selected as they contain hardware-related and software-related events. This study showed how these test executions behaved in virtual versus physical tests. The aspect of interest was the deviations between Fail test verdicts assigned to the executions. Beyond the analysis conducted to evaluate each approach of the test system, a questionnaire was performed to support the findings further and address the hybrid test system. Furthermore, FIT measurements were executed to estimate and measure how many failures (Invalid verdicts) occurred in each test system.

8.1 Findings Related to ReQ1

Each test system provided varying test results. Virtual systems were used more frequently for the testing. Meanwhile, physical systems started from a later stage of testing. Virtual systems were also the ones that executed more tests than physical systems. If only considering the quantitative data shown in figures 13 and 19-22 notably, virtual test systems provided more passing test results than physical test systems.

Before each test execution starts, all associated DUTs in each test system undergo a reset process to prevent side effects from earlier execution. We found that this process takes more time in physical systems than in virtual. Employees often face this type of challenge where timing differentiates between virtual systems and physical. Similarly, when a test requests ports or DUTs to link up or down, this process is also more delayed in physical systems compared to virtual. The cause of this could be traced to the nature of virtual systems. Identical findings were further supported by the questionnaire conducted. Furthermore, in the Related work section 3, we discussed the reality gap often encountered by practitioners in other fields [5, 58]. Therefore, the timing issue can be considered a real gap experienced at Westermo and the overall testing community.

Another challenge experienced at Westermo is that testing in virtual systems can sometimes mask hardware incompatibilities, leading to unreliable test results, creating false positivists. Timperley et al. in [51] described this challenge as part of the reality gap experienced by practitioners who uses virtual testing/simulations.

A third and more typical challenge is the lack of available devices when needed. Employees often struggle with finding devices as they spend more time searching and work gets halted. In some cases, developers and testers need more devices than accessible—this challenge results in complications, delays, and shortcuts in testing accuracy.

Even though our first research question aims to identify the differences between virtual and physical hardware, we must emphasize that few similarities between the hardware were also found. In Section 7.2 we analyzed how the Fail verdict differed in each test system approach and the potential reasons for the failures. However, we found that both approaches had very similar behavior during specific calendar intervals. For example, between the calendar intervals 16-25, test execution F3 was equally failure-prone in both hardware approaches. Similarly to execution F2 during the same calendar interval. Arguably, the virtual hardware used in virtual test systems replicates the physical entities convincingly. As reported in Section 2.7, the digital twin approach is a suitable way to describe the virtual test systems at Westermo.

The reliability measurement FIT enlightened the fact that physical test systems are often vulnerable to pure hardware failures. We found intervals where one physical DUT were unable to undergo...
a reset, which caused a barrage of failures in the test system. Test systems often need to be maintained to solve such issues, and there is always a risk that similar failures occur again. Similar issues did rarely occur in virtual test systems. Physical DUTs often require more attention than virtual ones. However, it is a logical problem, and Engblom et al. [17] highlighted the same issue. Engblom et al. further mentioned that sometimes testers or developers even had restricted access to physical hardware, which makes restart of devices or other desirable actions harder to perform. Virtual hardware (or simulated hardware in their context) typically solves these issues. However, FIT did further give the estimation that virtual test systems produce more failures (Invalid verdicts) compared to physical test systems. Most issues in virtual test systems were related to link-up/link-down of ports, failures in test implementation, or test framework, meaning that the adversity of losing a DUT during testing are possible to overcome.

8.2 Findings Related to ReQ2

Employees who answered the questionnaire had similar opinions about a hybrid test system, both those that already use it regularly and employees who see an advantage of having a hybrid test system available. Section 7.2 highlighted that test execution F2 is not dependent on hardware features. Therefore, F2 is an example of an execution that could be executed in the hybrid test system without compromising the reliability of test results. Furthermore, hardware-specific cases could be covered by the hybrid system. This is a significant advantage that could be collected from a hybrid test system.

Earlier research papers discussed in Related work 3 did either recommend a transition from virtual testing to physical testing [48] or using an approach that combines both virtual and physical hardware in one entity (i.e., a hybrid system) [21, 58]. However, the recommendation from Tahera et al. in [48] is already accomplished by Westermo. Meanwhile, Himmler’s and Wilkinson’s [21, 58] recommendations should be considered more profound.

The questionnaire analysis highlighted that GNS3 was the popular simulator used by the employees to implement a hybrid system. GNS3 uses QEMU as an emulator to emulate the physical devices. One advantage of using GNS3 is bypassing much QEMU configuration. However, it has not been systematically used to run automated testing at Westermo. A hybrid system implemented in GNS3 has been used by some of the employees to achieve efficiency when there was a lack of enough physical hardware in their desktop setups.

To achieve automated testing using an external simulator, a third-party framework tool must participate and act as a communication link between the simulator and the environment where physical devices are implemented. In this case, three different tools must cooperate to build a hybrid system. The risk with cooperating multiple frameworks is creating new error sources.

As an example, the hybrid system is beneficial in the software development process, such as during nightly testing. The test system could be incorporated into the testing phase and used as a reference system until pure physical systems are available. For example, a software branch could firstly be tested completely virtual. At this stage, build-verification testing could be run on the virtual systems. The hybrid system could be used once at least one physical hardware is available, see Figure 14 for illustration. As a hybrid system allows the combination of both physical hardware and software, typical testing at this stage would involve hardware-specific cases. As soon as pure physical systems are available, the testing could be shifted towards them. This approach takes advantage of all testing systems options, testing with physical hardware early and testing from early development phases.

Another possible usage scenario where the hybrid system could be beneficial is when testing customer-specific cases, such as for the customer support department. A customer may have a setup of many more DUTs than in the test systems, where most of them run older versions of WeOS and some of the DUTs runs the newest versions of WeOS. In Section 7.4, the questionnaire analysis highlighted that virtual systems only run the latest version of WeOS, meaning that to test the older versions, physical systems must be initiated. In this example, a hybrid system could overcome two challenges and add flexibility by replicating the number of devices used by the customer and running multiple operating system versions in one closed entity. The physical DUTs in
the hybrid system could be used to replicate the customer setup of older devices and all virtual DUTs to replicate the newer versions of devices.

Figure 14 demonstrates a hypothetical example of how a hybrid test system could be built. In this example, four DUTs D1-D4 forms a ring. D1 is a Physical DUT and D2-D4 are emulated DUTs inside a PC. Link breakers are not considered in this case.

The example illustrates that D1 is physically connected to the PC using two ports, one for input and the second one for output. The physical input link is further divided into four virtual links, three of them connected to D2 to build a ring between the devices. Also, D4’s output is three virtual links combined as one physical output link to D1. D1-D3 and D2-D4 are connected cross-like.

Beyond the advantages mentioned above, the hybrid test system could further arise with the threat of introducing new timing issues or new fault masking. The answers from the questionnaire indicated how virtual systems sometimes mask faults, typically hardware incompatibilities. This type of threat may be introduced in the hybrid system since virtual hardware will play a significant role in hybrid systems. Fault masking is a vulnerable point that should be considered when running tests on such systems. The hybrid system should be evaluated from the early stages by running the same tests on all three versions of the test system to observe differences. The evaluation could contribute to which type of test fits the hybrid system without compromising the beneficial attributes of such a system. F2 is a test case that is recommended by this thesis to execute in the hybrid system.

### 8.3 General Findings Unrelated to Research Questions

Earlier in this thesis, we discussed the state of practice in virtual/physical testing or virtual/physical hardware used for software testing. We found several approaches addressing the same objective as in our case. The approaches are XiL, MiL, HiL, SiL and Digital Twin, they are described in the Background sections 2.6 and 2.7. We start by brushing up our memory and recalling what these approaches mean:

**X-in-the-Loop (XiL)** is the main approach that is widely used in different industries. Meanwhile the remaining approaches MiL, HiL and SiL are branches derived from the XiL. The in-the-Loop term indicates the relationship between the system itself and the operational environment.

**Model-in-the-Loop (MiL)** is an approach often used when practitioners have a model of the operational environment and a model of the physical hardware. According to Figure 10, MiL is used during the requirements and design establishment phases.

Once the initial software implementations are ready to be tested, **Software-in-the-Loop (SiL)** can be used for software verification purposes. According to Table 1, hardware and the operational environment are still virtual at this stage.
**Hardware-in-the-Loop (HiL)** provides the combination of both physical hardware and virtual testing. HiL is often used for system-level testing. At this stage, the physical hardware is available for testing. Meanwhile, the operational environment is virtual.

The **Digital Twin** approach is another well-established approach aiming to replicate a physical system digitally. For example, it has been used by NASA’s Apollo space program as they built two identical space vehicles. The replicate helped in simulation and predicting the conditions of the vehicle on the mission.

We learned from these approaches that simulations are widely used in many industries. We found a lot of papers describing the HiL, SiL and Digital Twin in the automotive, space, and aviation industries. However, finding research more oriented towards networking devices, software engineering, or test systems dedicated to software testing was harder. Therefore, these findings had to be generalized to fit the context of this thesis.

The typical terminology used at Westermo has always been "physical test systems," "virtual test systems," or "physical/virtual testing." After the knowledge gained from Sections 2.6 and 2.7, following conclusions can be made:

- The virtual testing/virtual test systems are the **SiL** testing approach
- The physical testing/physical test systems are the **HiL** testing approach
- The hybrid test system is a mixture of HiL and SiL. Therefore the terminology **Hybrid-in-the-Loop** is recommended

**Software Reliability**

The software under test’s (i.e., the selected branch) reliability has not been considered in this thesis work. Instead, this thesis have targeted differences between virtual/physical hardware used for the testing and possible benefits from a third option test system. However, we found an interesting aspect worth mentioning. In Related work 3 subsection **Software Reliability**, we discussed the Figure 11 which contains the Idealized curve and Actual curve of a software development. This Figure is comparable to Figure 13. We observed a decrease in the failure rate from the middle of the branch timeline and towards the end when investigating the selected branch. In the early stages of the branch timeline, many failures (Fail verdicts) were discovered and attended to, resulting in fewer failures towards the end of the branch. Arguably, developers may have new requirements to consider, new functionalities to implement, or, typically, faults in the coding phase. However, these events caused a high rate of failure. The failure rate decreased once they were attended to, and a healthy branch with an acceptable failure rate level was obtained.

**8.4 Validity Threats**

This subsection contains a discussion of possible validity threats.

When analyzing the Fail verdict, only one branch was selected, and all archival data were collected from that branch only. Even though the selected branch was broad enough and involved an equal number of test systems of both versions (physical and virtual), it is arguable that multiple branches should be considered to gather further understanding and diversity of results. For the case of this thesis, due to time limitations and since two data collection methodologies were used, investigating one branch was enough and hinted valuable results for the company of interest.

The second possible threat to the validity is the FIT measurement presented in section 7.3. Researches from Pressman and Maxim [33] and Wood [61] described how to perform the measurement and estimate the time factor. Further, the Invalid verdict was selected as a state where the test system failed to deliver the expected functionality. Other types of measurement could have been applied to evaluate the reliability of test systems. For example, how often does a test system totally fails to assign any verdict to an execution.
9 Conclusions and Future Work

Physical hardware dedicated to testing has been the go-to approach by many practitioners in several industries. However, having physical hardware dedicated for testing is not always sufficient due to economic aspects, as hardware preparations may take more time than expected or get damaged during testing. Virtual hardware is one solution that can facilitate testing and reduce the usage of physical hardware.

In this master’s thesis, conducted at Westermo AB, we compared the software testing in physical and virtual test systems and evaluated possible benefits from a hybrid test system consisting of virtual and physical hardware.

Westermo specializes in designing and manufacturing robust data communication devices for mission-critical applications. A typical device produced by Westermo is an Ethernet Switch, and a typical customer is the rail industry. A test system, in the context of Westermo, is an entity that contains Devices Under Test (DUTs), links that connect the devices and nodes to illustrate different scenarios which may encounter in the field. Test systems are available in two established versions, physical and virtual, see Figures 5 and 6 for further illustration. The virtual version of a test system is built similarly architectural-wise, but all devices, links, and nodes are emulated virtually.

ReQ 1: What are the main differences and what challenges are experienced when virtual hardware and physical hardware are used for testing embedded systems software?

In terms of differences between virtual and physical hardware used for testing, this research highlighted that the reset process of DUTs faster in physical hardware, but ports' link-up/link-down time is slower in physical devices.

Another difference found is that physical test systems often fail due to power losses in DUTs. Similar issues rarely occur in virtual test systems.

The third difference concerning one of the test executions investigated, F1, which is undesired access to the local network and devices. This specific execution had better Pass-run in virtual systems than physical but was often more time-consuming (taking more time from execution until a verdict is assigned).

Considering the Failure In Time (FIT) measurement, one of the physical test systems (Physical4) was the most failure-prone test system. Most failures found were due to DUTs not being able to undergo a reset when needed. When comparing with virtual test systems, virtual ones often had issues with failed link-up/link-down of ports.

A challenge often experienced by employees at Westermo is that physical hardware is a restricted object, often leading to time-wasting while waiting for hardware to be available. Sometimes testing of customer-related cases requires more hardware than accessible, which causes delays and shortcuts in testing accuracy.

Another challenge expressed by employees is the "False Positive" in virtual systems. This challenge causes a reality gap between virtual and physical hardware. Virtual systems do not always spot hardware incompatibilities and cause better test results than expected. Another challenge related to the reality gap is timing issues often masked in virtual tests but discovered in physical testing.

ReQ2: What benefits can be provided from a hybrid test system consisting of virtual and physical hardware?

A hybrid test system, consisting of both virtual and physical hardware combined in one system, can provide benefits such as an expansion of the already existing test systems, testing for hardware-specific cases and regular testing simultaneously, and testing multiple versions of WeOS at the same time. Beyond these possible benefits, the reality gap and timing issues experienced in virtual/physical hardware may be overcome with the hybrid system.
9.1 Future Work

This thesis has given insights that a hybrid test system may be beneficial in numerous ways. The hybrid system could either be implemented in the existing test framework of Westermo or by using a commercial platform such as GNS3. It could help solve virtual test systems’ challenges and issues experienced with both options of test systems. Further, one could analyze the reliability of each test system approach by executing the same test simultaneously and observing the outputs and logs. It is worth inspecting if the reality gap and timing issues decrease. For that matter, a proof of concept implementation can extend this thesis. Another possible future work in the same direction as this thesis is investigating several branches to observe diversity between test verdicts or text executions.
References


[57] Westermo. WeOS user guide, 2021.


A Questionnaire about Test Systems used for Embedded System Testing at Westermo

This survey is part of a thesis work, which aims to evaluate the virtual and physical test systems. Thank you for your contribution, all answers will be anonymous. The survey will take approximately 45-60 minutes to complete.

- **Section 1:**
  - Q1: How long have you been working with embedded system testing?
  - Q2: What is your current position at Westermo? Example: Software developer, Test framework developer, support.. 
  - Q3: Are you using any of the virtual/physical test systems on daily basis? If yes, which one do you use more frequently?
  - Q4: How many physical DUTs do you need for your daily work?
  - Q5: Have you encountered the problem where physical DUTs are not available when you needed them?
  - Q6: What consequences have you experienced in your daily work as a result of physical DUTs not being available?
  - Q7: What are the typical issues you face when trying to use virtual test system?
  - Q8: What are the typical issues you face when trying to use physical test system?
  - Q9: Which type of test systems (virtual or physical) covers your needs mostly during your daily work? And why?
  - Q10: Based on your experience, what are the main benefits and drawbacks with the physical test systems?
  - Q11: Based on your experience, what are the main benefits and drawbacks with the virtual test systems?
  - Q12: Which of the test systems (virtual or physical) are most reliable in terms of test results, based on your experience? And why?
  - Q13: What type of tests would you prefer to do in virtual test systems/physical test systems? And why?
  - Q14: If hybrid test system (virtual and physical DUTs) were available, how would such test system contribute to your daily work?
  - Q15: How reliable would the test results be in a hybrid test system, when compared to virtual and physical test systems? And why?

- **Section 2:** For you which have implemented a hybrid test system
  - Q16: How did you implement the hybrid test system?
  - Q17: Why did you need to implement a hybrid test system?
  - Q18: How many physical and virtual DUTs did you have in your hybrid test system?
  - Q19: What was most challenging in implementing the hybrid system?
  - Q20: Which tests did you execute on the hybrid system? And did you re-run the tests on virtual/physical tests system for confirmation?
  - Q21: Which differences did you encounter running tests on the hybrid system vs virtual/physical test systems?
  - Q22: What are your experiences running automated testing on the hybrid system? Did you face any specific challenges?
B Invalid Verdicts

The figures 15 and 16 provide an overview of how the Invalid verdicts differed in each test system during its running time. It is observable that test systems were used at different calendar intervals. However, the Y-axis in all figures presents the number of Invalid verdicts, and the X-axis illustrates which nights the test system was active for the software testing.

![Graphs showing invalid verdicts for each physical test system and the nights they were running](image)

Figure 15: Invalid verdicts for each physical test system and the nights they were running
Figure 16: Invalid verdicts for each virtual test system and the nights they were running
C  Test Verdicts Overview for each Test System

The figures 17 and 18 presents the quantitative data for each test system over the entire branch timeline. The verdicts were selected to analyze and compare Pass, Fail, and Invalid. These verdicts are further discussed in Section 7. Moreover, these figures further illustrate the distribution of verdicts.

Figure 17: Pie charts illustrates the number of Pass, Fail, and Invalid verdicts for all virtual test systems
Figure 18: Pie charts illustrates the number of Pass, Fail, and Invalid verdicts for all physical test systems.
D  Test Verdicts Overview over each Nightly Testing Session

This appendix contains bar charts which in detail illustrate the distribution of verdicts, which were assigned during the testing, for each test system. The X-axis in all figures below represents the test days, i.e., from the first night, the test system was used until the last night. Meanwhile, Y-axis is the number of test verdicts each nightly session. The scale in the Y-axis are varying depending on how many verdicts were assigned each night.

![Physical2 Test Verdicts](image1)

![Physical1 Test Verdicts](image2)

Figure 19: Bar charts illustrating the distribution of verdicts for each test system over the nights it was used for testing
Figure 20: Bar charts illustrating the distribution of verdicts for each test system over the nights it was used for testing.
Figure 21: Bar charts illustrating the distribution of verdicts for each test system over the nights it was used for testing.
Figure 22: Bar charts illustrating the distribution of verdicts for each test system over the nights it was used for testing.