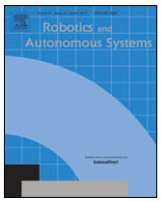
Robotics and Autonomous Systems 165 (2023) 104424

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot



An exploratory study of software engineering in heavy-duty mobile machine automation

Andrei Ahonen [a,*], Marea de Koning [a], Tyrone Machado [a,b], Reza Ghabcheloo [a], Outi Sievi-Korte [c]

[a] Faculty of Engineering and Natural Sciences, Tampere University, Korkeakoulunkatu 6, Tampere, 33720, Finland
[b] Bosch Rexroth AG, Department of System Development, Elchingen, Germany
[c] CSC – IT Center for Science Ltd., Keilaranta 14, Espoo, 02150, Finland

ARTICLE INFO

Article history:
Available online 10 April 2023

Keywords:
Machines
Heavy equipment
Cyber–physical systems
Software engineering
Robotics
Industry

ABSTRACT

As the amount and complexity of software for automating heavy-duty mobile machinery is increasing, software engineering in this domain is becoming more important. To characterize the industry's current state of software engineering and its issues to guide future research, we performed an empirical exploratory study. We interviewed 16 software engineering professionals from 13 different companies conducting business in heavy-duty mobile machines and their automation. The interviews were analyzed qualitatively, and quantification of the analysis results is presented. We first create an overview of software engineering in the heavy-duty mobile machinery industry. We then identify problem areas affecting software development and discuss some of the possible solutions found in literature. Our findings indicate that the major problem areas faced in the industry that require more research are its digital transformation, autonomous machine functional safety, low availability of workforce for developing software for robotic mobile machines and the lack of established software standards.

1. Introduction and related work

1.1. Introduction

Heavy-duty mobile machines (HDMM) are machines used in industries, such as construction, earth-moving, agriculture, forestry, ports, and warehouses. They are a subset of industrial machinery and they are also known as mobile working machines (MWM), non-road mobile machines (NRMM), non-road vehicles, or heavy equipment. While these machines are still predominantly manually operated, they are being automated at an increasing pace. The new automated functions require complex software running on PC-like computational platforms, which are currently only sparsely utilized on the machinery. This swift utilization of PC-level hardware has led the HDMM industry to use software engineering practices and professionals, but it has been unclear how the latest knowledge of software engineering, having origins in non-machinery applications, fits into the more traditional engineering-focused HDMM industry. Most of the HDMM automation-specific literature focuses on more mechanical or physical topics, such as powertrain automation, actuator improvement, and energy efficiency. There has not yet been a widespread effort to study the more abstract aspects of HDMM automation. This study aims to create an overview of the software engineering contexts and practices in automating HDMM, and the consequent problems encountered by the HDMM industry.

In addition to the general need for technology transfer between academia and industry, the lack of HDMM-specific, industrially relevant knowledge on this topic forms the basis of our motivation to study HDMM-specific software engineering in the industrial context. The main contribution of this study is that we provide an empirical view of software engineering specifically for the HDMM industry. The article is structured as follows: first, we define the research questions, describe the data, its collection, and subsequent analysis in Section 2. Section 3 provides the results of the qualitative data analysis. After these results, the research questions are answered in Section 4. The threats to validity are discussed in Section 5 which is followed by the final conclusions in Section 6.

1.2. Related work

To the best of our knowledge, there exists only a limited number of empirical studies with industrial participants on software

* Corresponding author.
E-mail addresses: andrei.ahonen@tuni.fi (A. Ahonen), marea.dekoning@tuni.fi (M. de Koning).

https://doi.org/10.1016/j.robot.2023.104424

engineering for cyber–physical systems.[1] One such exploratory study on software engineering problems for cyber–physical production systems is from Feichtinger et al. [2]. They conducted a workshop with academic and industrial participants and present 8 problem areas in their article: complexity, variability, real-time requirements, knowledge management, multidisciplinarity, agility, education, data analytics, and AI. Many of these topics are similar or have connections to the themes we describe later in this article. For example, we discuss their topics of multidisciplinarity, education, and organizational knowledge with slightly different terminology, mostly under the term *human capital*. Also, Feichtinger et al. [2] present multiple recommendations for future cyber–physical production systems software research, such as developing tools for improved interdisciplinarity and product variability management. From the point of the HDMM industry, their study is helpful as industrial automation can often be utilized on automated mobile machinery. Our study sheds some light on the HDMM side of the topic, and can hopefully be later used to further establish the differences and similarities between manufacturing automation software and HDMM automation software.

In a similar study in service robotics [3], an empirical survey was conducted that combined interviews and a widely distributed questionnaire. They reached out to both academic and industrial practitioners of robotics, thereby gaining a strong empirical foundation for their findings. They observed, *inter alia*, that there are no standards or best practices yet in robotics software development and that the domain requires high error tolerance from the runtime software of robots.

A general article on cyber–physical systems software engineering [4] discusses CPS applications, differences to general software engineering, and the issues faced in CPS software engineering. The applications listed in it are medical systems, smart buildings, smart grids, oil and gas pipelines, smart water networks, vehicular safety systems, manufacturing systems, and autonomous driving. The differences to general SE are listed as real-time behavior, heterogeneous hardware, distributed computation, security and privacy, reliability, large-scale and real-time communication needs, support for mobility, power limitations, integrations with other systems, and context awareness. As for the software engineering issues, the authors identify the following areas: requirements engineering, modeling and simulation, CPS middleware, development tools, testing and verification, and education. The main shortcoming of the article is that, while very salient otherwise, it does not seem to be directly based on empirical evidence. However, many of their findings are repeated by other literature which mitigates this issue, and our findings are also in consensus with a subset of their arguments.

The general topic has been explored also in multiple workshops that have been arranged as a part of the International Conference on Software Engineering (ICSE). The first workshop [5] identified the following issues that need further research: the role of humans in the systems, multidisciplinarity, and uncertainty in CPS. The second workshop [6] identified issues relating to security, software engineering processes, and emergent designs. The third workshop [7] discussed open problems on CPS modeling, system boundaries, and human-centric non-technical aspects. The fourth workshop [8] brought attention to issues on metamodeling and modeling, autonomous CPS, and the importance of good reference cases. The workshop participants were both from academia and industry, but due to the generality of their topic, it is difficult to gauge how the issues raised during the workshops relate to the HDMM industry.

[1] Definition of cyber–physical systems from Zheng and Julien [1]: "*Cyber–Physical Systems (CPS) are an integration of computation and physical processes*".

Considering HDMM as a technical domain in itself, a comprehensive definition and treatise are presented by Geimer in "*Mobile Working Machines*" [9], in which the HDMM are referred to as mobile working machines (MWM). For the purposes of this study, the terms HDMM and MWM are understood to be equivalent. This study aims to supplement especially knowledge on machine control [9] since only a limited consideration is given to software-related topics. Also the focus on machine control in the work of Geimer [9] is on the hardware level, such as communication networks and sensors.

Robotics, especially mobile and field robotics, shares many technical features with HDMM. However, HDMM applications are less automated. According to ISO standard 8373:2021[10], the term *robot* implies autonomy. Standardized definitions for the levels of autonomy in HDMM are still being developed [11], but in general, the level of autonomy for HDMM is still lower than that of robots, because the HDMMs are designed for human operators or drivers. Using the ISO 8373 vocabulary, one could refer to a non-automated HDMM as a *mobile platform*, partially automated HDMM as a *robotic device* and driverless HDMM as a *mobile robot*.

We claim, however, that there remain two loosely defined contextual differences between robots and HDMM. The first difference is the history of the HDMM industry and the role of its products in operational industries. HDMM have been produced for over a century now [12], thereby having established supply chains, business models, and processes. Thus, HDMMs are heavily embedded in their operational industries. By being embedded, we mean that HDMMs are not usually considered as a technology in itself, but rather as tools in the operational industries. This embedding can be seen in ISO standards classification scheme for machinery in operational industries such as agriculture [13], construction [14], materials handling, earthmoving [15] and mining [16]. Each category has HDMM, but they are included in the ISO catalogs with other, non-mobile operational industry equipment. In comparison, robotics is still an emerging field without heavy conceptual ties to other industries outside of manufacturing. This can also be seen in ISO standard catalogue classification for robotics, where it is listed under **manufacturing** [17].

The second difference is the size and power of the machines, i.e., they introduce different kinds of safety hazards which reveal different safety requirements and are more expensive than robots. Driverless HDMM poses much greater physical hazards than comparatively small service robots and they consume far more resources, such as energy, space, and human attention. The HDMMs have to be powerful in order to achieve their intended purpose: high throughput manipulation of the environment. This also means that not all safety strategies viable for robotics can be applied in HDMM, *i.e.* the HDMM cannot be made smaller, slower, or softer like robots can be. Furthermore, in practice, HDMM can be differentiated from robots by their power transmission systems. Robots usually operate using electro-mechanical systems such as electric motors and mechanical transmission, while HDMM utilizes fluid power or hydraulics due to its mechanical advantages such as compact size, and high power and force densities [18] over other power transmission technologies.

Another similar technical field is autonomous road vehicles. An extensive literature study has been done in the field of automotive software engineering [19]. However, road vehicles operations consist primarily of driving/navigating/propulsion from one location to another, without manipulating (changing the shape, size, form, and/or location) external objects, whereas, for HDMM, both manipulation and driving may play a significant role in the operations [11]. This is reflected in the requirements of the automation software. Some of the autonomous driving innovations and technologies transfer well into the navigational and base propulsion tasks of mobile machinery, even though the manipulation tasks may need to be handled differently [11].

Table 1
Interviewees.

Person ID	Company ID	Years in HDMM industry[a]	Educational background	Currently works with
1	A	2	N/A	Hardware + software
2	B	8	CS	Software
3	B	12	Automation	Software
4	C	N/A	Electronics	Software
5	D	2	Automation	Software
6	E	18	Automation	Software
7	F	1	CS	Software
8	G	3	Automation	Software
9	H	7	Automation	Software
10	I	14	Embedded systems	Software
11	J	5	Automation	Software
12	K	5	Marine engineering	Hardware + software
13	B	15	CS	Software
14	M	21	N/A	Hardware + software
15	L	11	Mechanical engineering	Hardware + software
16	F	4	CS	Software

[a]Only in professional role, many interviewees had HDMM experience in non-professional contexts. Many interviewees also have numbers of years of experience in other industries and academia, which are not counted here.

2. Research process

In this section, we start by outlining the research setting and our research questions, and then explain the interview setting.

2.1. Research setting

In the classification of empirical strategies [20], this study is an *explorative survey* with qualitative data collected from *unstructured* interviews. The interview data was analyzed using *thematic analysis* [21]. Further methodological details can be found in Appendix A.1.

To clarify the current state of software and its production in the HDMM industry, we specify three research questions (*RQ*). The questions are meant to be broad to explore more than to explain.

RQ1: How and what kind of software is produced in the industry for HDMM automation?
The answers to this question provide the necessary background information for the following research questions.

RQ2: How are HDMM software and its production different than those from general IT?
The main motivation behind comparing HDMM with general IT software systems is the availability of human capital and a wider body of literature. By general IT, we mean information systems that are not cyber–physical systems or software that does not handle measurement signals or command actuators. This general IT could mean healthcare systems, databases, web services, content management systems, office automation, or any other system where the goal is to control and manipulate information ultimately only for human consumption. This is contrasted with cyber–physical systems where the information is also consumed by controlling machinery without human intervention.

Answers to this research question can be used to define the domain-specific sections and use that information to focus future research efforts on the topic. On the less domain-specific sections, it should be easier to utilize research results found in the general information systems context.

The answers come with a caveat as this is an exploratory study, aiming not to confirm hypotheses, but to generate them. Thus, all the results gained from this study, especially for this research question, should be considered hypothetical until further evidence on the topic is presented. On the other hand, this study also aims at building theory through *inductive* reasoning based on collected data as described by Eriksson & Kovalainen [22].

RQ3: What are the problems or limiting factors faced in the industry when developing HDMM software?
Answers to this research question will help focus future research and aid the technology transfer between academia and industry. Especially so if the answers are combined with knowledge about domain-specific aspects gained from the previous research question.

2.2. Interviews

The interviewees were selected through a two-phase process. In the first phase, suitable companies were selected based on networks formed from previous university-industry collaborations. The second phase of the selection was done by each company based on the invitation (see Appendix A.2 for the whole invitation), where we specified: "*The ideal interviewee would have a great deal of experience in developing software for mobile machines.*". So if the company decided to participate in the study, they would pick the person to be interviewed. The invited companies were HDMM manufacturers, system integrators, and technology providers working with HDMM and their automation. All of the companies are based in Europe, but most of them do business globally. The general invitation was sent to 19 companies, 13 of which responded, and an interview or interviews with each of the 13 companies were scheduled. The interviews were conducted from April to June 2021. Information on the interviewees can be found in Table 1. The individuals or companies will not be named here due to confidentiality.

The discussion was steered with an interview guide, which was followed unless the interviewee wanted to explore other topics they deemed relevant to the topic of software development for automating mobile machinery. Thus, the interviewees had much control over the discussion, and the interview guide was used to formulate questions when the interviewee had no particular interest in discussing something. The interview guide shown in Appendix A.2, is a list of specific topic names, partitioned into three themes based on the research questions, with each having a separate subtopic as a basis for a possible question. The interview guide can be summarized here as follows:

Overview of current state of software development in the HDMM industry: *Interviewee background and experience, overview of the software lifecycle, tools, methodologies, frameworks and paradigms*

Domain specificity: *Typical subsystems, components and hardware, simulation and testing, necessary domain knowledge, comparison of this domain to others*

Challenges: *Limiting factors, time consuming activities, failures, successes*

Table 2
The list of themes from analyzing the interviews.

Code	Quotations	Addressed RQ:s
Difference to IT	16	RQ 2
Human capital and workforce	47	RQ 1, RQ 3
Level of autonomy	13	RQ 1
Management, organization and business	68	RQ 1, RQ 3
Ongoing industry digital transformation	13	RQ 1, RQ 3
Programming tools and model-based engineering	27	RQ 1
Releases and configuration	21	RQ 1
Safety	34	RQ 3
Software design and codebase evolution	26	RQ 1
Runtime software functionality, structure and underlying hardware	53	RQ 1
System integration and lack of standards	7	RQ 3
Testing and simulation	48	RQ 1, RQ3

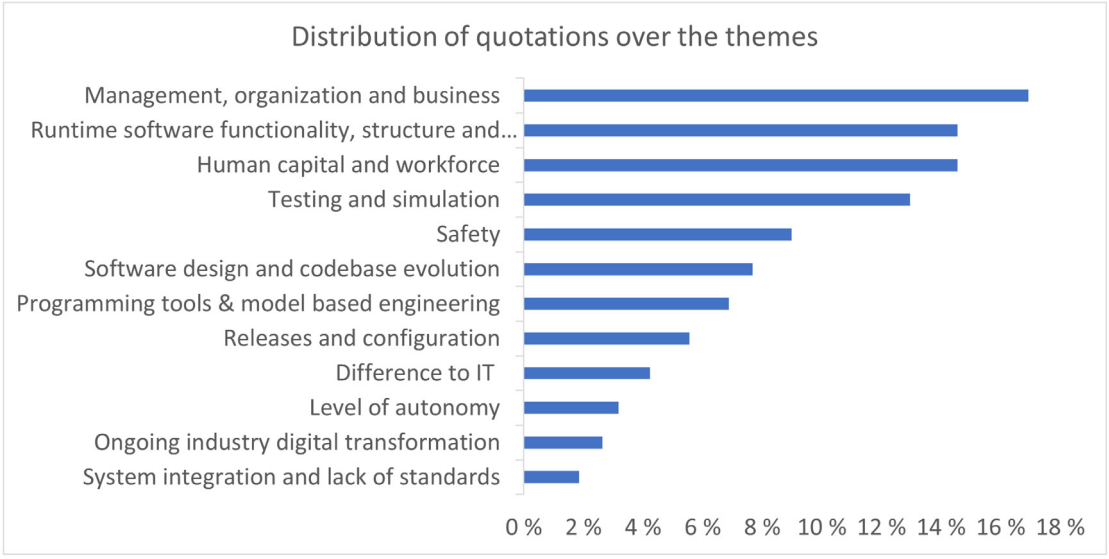


Fig. 1. How the numbers of quotations coded under a theme are distributed.

The interviews were conducted, recorded, transcribed, and anonymized by the first author for analysis. Only the first author had access to full interview material. The anonymized data were also available only to the first, fourth, and fifth authors due to confidentiality and ethical guidelines. Two particular interviews were conducted as 1-on-2 discussions such that two employees of a company participated simultaneously with one individual (person ID 13 in Table 1) present in both interview sessions. So from a single company total of three different people were interviewed. Most of the interviews were done in Finnish, and a smaller fraction was done in English. Detailed information on the interviewees is found in Table 1. More information also on the interview transcriptions is shown in Appendix A.4.

3. Qualitative analysis findings

The themes (codes) gained from the analysis are summarized in Table 2. For each theme listed in it, we give the number of quotations coded under that theme and the RQ:s that the quotations relate to. More accurate answering to the RQ:s will be left for Section 4, where we synthesize the answers from the observations. Here the mentions of touched RQ:s are only meant to give an overview of possible discussion content before looking at observations more closely. We will further describe each theme and possible overlaps with the other themes, and we will synthesize the most relevant observations and aspects from the interview data in a few paragraphs under "*Main observations*" title. Some themes have an example quote from the interview data. The more fragmented observations are collected to Appendix B.

Figs. 1–3 depict a quantification scheme in order to emphasize different ways to view the results. One way is to simply look at the distribution of the number of quotations over the themes in Fig. 1. It shows how four of the themes account for roughly 60% over all the quotations.

The interviewees had varying levels of experience in the HDMM industry. In order to see how the experience influenced the discussion, the quotations were weighed based on the interviewee's experience and the result is shown in Fig. 2. The formula for weighting the quotation with experience is $E \times Q$, where E is the experience of the interviewee in years in the HDMM industry and Q is the number of quotations under a specific theme from that person, *e.g.*, 1 quotation coded under "*Safety*" from a person with 4 years of experience counts as 4 quotations coded under "*Safety*".

As a result of this weighting, the results change slightly. The difference between the weighed and non-weighed distributions is shown in Fig. 3. This figure shows the percentage point difference for each theme's share of the quotations and from it, we can see that the more experience the interviewee had, the more the quotations skew towards some topics and away from others.

We also quantified how many participants supported the themes and their main observations. Fig. 4 presents how many participant's provided quotations for each theme. To quantify the number of participants who supported the main observations made under each theme, we screened the interviews and identified whether the participant's quotations overall relates to and supports the main observations. Fig. 5 shows the results.

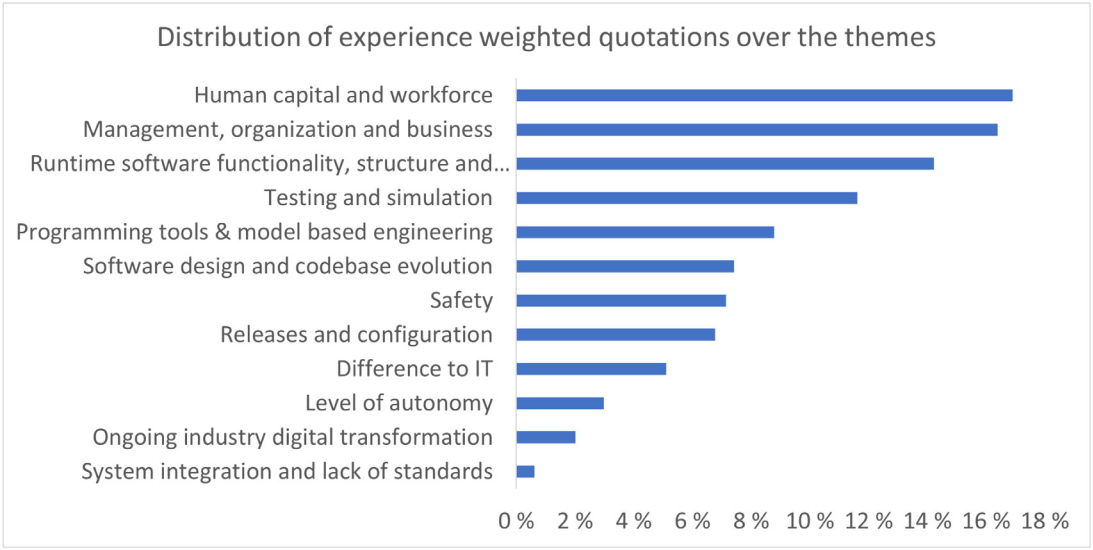


Fig. 2. Experience weighted distribution of quotations over the themes.

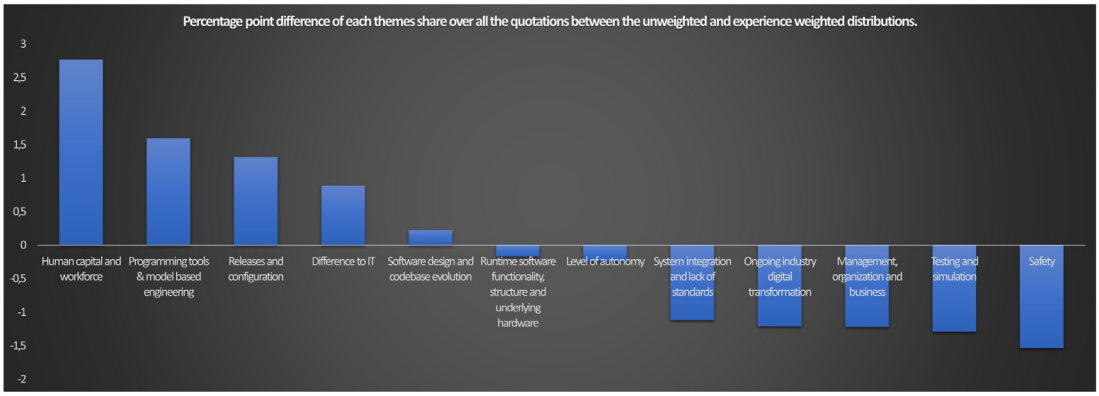


Fig. 3. Percentage point differences of each theme's share over all the quotations between unweighted and experience-weighted distributions.

These quantifications can be compared and we can see how the proportion between the number of contributing participants for both cases is roughly similar. The three most widely supported themes keep their relative place when looking at their contribution to the main observations and the three least supported themes still stay at the bottom. The numbers of people contributing to the main observations for each theme are smaller, which is natural as quotation content was split into two groups, *main observations* in Section 3 and *fragmented observations* in Appendix B.

We will next have a closer look into the themes listed in Table 2.

3.1. Theme: Difference to IT

The observations that belong to the theme "*Difference to IT*" are based on the quotes that describe how mobile machines as a technical environment and the software development for them (HDMM) differ from those in general information systems.

Main observations:

All programs manipulate and route data in some form. In information systems programming, the data hold exact information such as money or health information. The data flows from one API to another without many corner cases or strict temporal requirements and concurrency can often be avoided. This makes it comparatively easier to reason about program behavior.

In HDMM automation software, the sensor-based data are not exact as it contains noise, the program often has to be executed in real-time, there are many more corner cases and concurrency is unavoidable. The vast amount of corner cases compels the runtime software to be robust to them.

3.2. Theme: Human capital and workforce

The term *human capital* [23] is a wide concept, but here it is a label to address the skills and knowledge a software developer might have and how this knowledge affects software development. This could mean either software engineering skills and knowledge, such as that contained in Guide to the Software Engineering Body of Knowledge [24], the domain knowledge of HDMM [9], embedded systems and physical modeling such as in Lee and Seshia [25], network protocols or knowledge of the operational industry.

The observations under the theme "*Human capital and workforce*" are based on quotes about how human capital and available workforce affects software development for HDMM automation in general.

Main observations:

HDMM software development suffers from a lack of suitable workforce. Very rarely are developers simultaneously highly skilled in programming, have sufficient domain knowledge of HDMM or its operational industry, and be motivated to work on

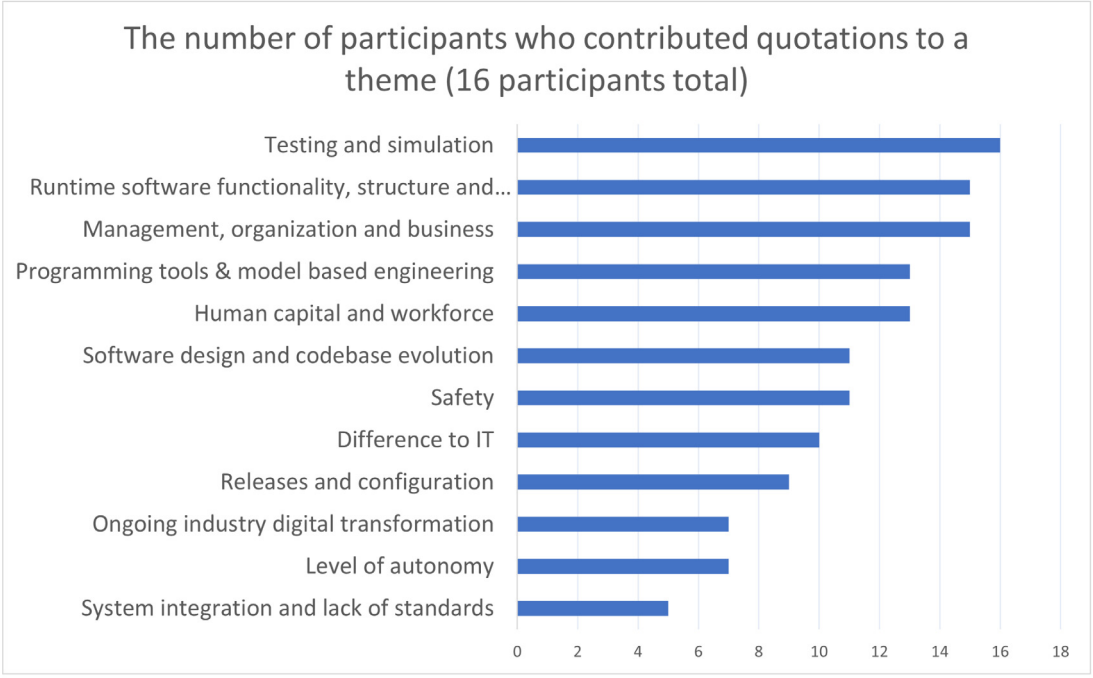


Fig. 4. The number of participants whose quotations contributed to a theme. Horizontal axis shows the number of participants and the themes are listed in the vertical axis. For example, you can see from it that 10 (out of 16) participants provided one or more quotations that were classified under the theme "*Difference to IT*".

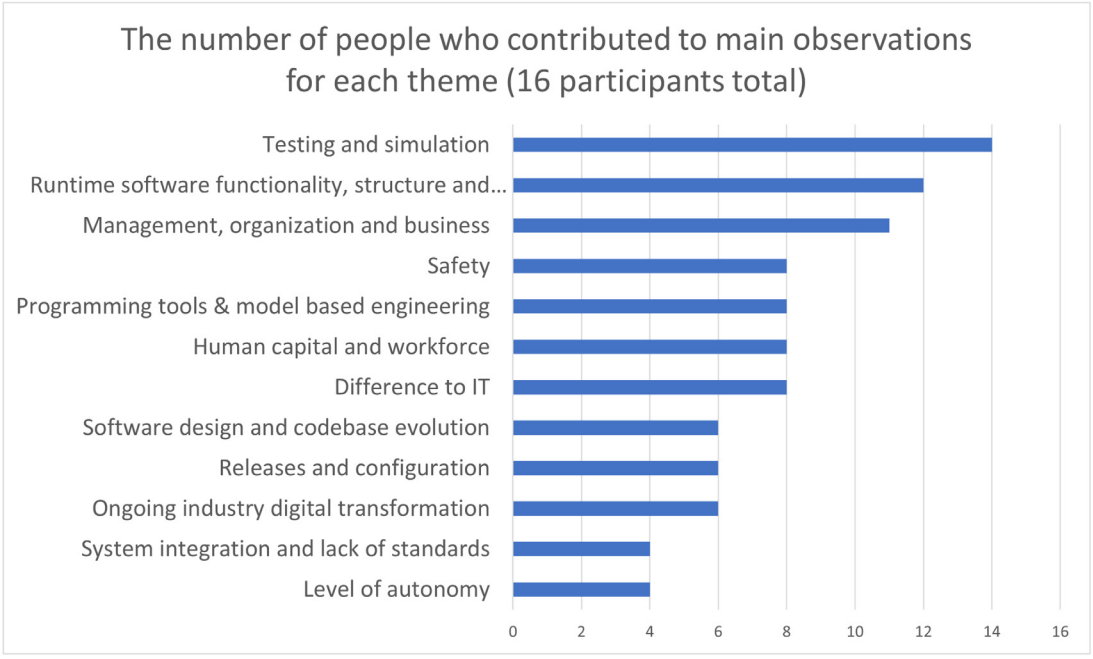


Fig. 5. The number of participants whose quotations contributed to the ***main observations*** for each theme. Horizontal axis shows the number of participants and the themes are listed in the vertical axis. For example, you can see from this figure that 4 (out of 16) participants provided one or more quotations that were classified under the theme "*Safety*", and which also contributed information to the "*Main observations:*" part in Section 3.9.

the system. Even if an employer finds a new suitable developer, they still have to learn the codebase before they can actually contribute to it, which can take up to a year. These notions are exemplified with the following quote: "*Probably one of the most hindering factors is the lack of skilled and motivated workforce. By motivated, I mean wanting to do this type of work, which is... Well if we say that reaching a reasonable level takes half a year, so it is pretty long-term work.*"[2] Junior developers also often lack experience especially in what is known as "*programming in the large*" [26].

When speaking of workforce skills in general, critical domain knowledge is utilized in software production in at least three

[2] Quotation translated from Finnish by the first author.

ways. First, the software engineer may have and use the necessary domain knowledge to interpret ambiguous requirements. Second, unambiguous requirements are formed by a domain expert using the necessary knowledge. Third, models may be created by domain experts that can be used to generate source code. (Conceptual overlap with "*Programming tools and model-based engineering*".)

3.3. Theme: Level of autonomy

The theme "*Level of autonomy*" includes the observations from the quotations mentioning different levels of autonomy in HDMM and how software relates to it.

Main observations:

Many automated HDMM still require intermittent human intervention, for example when the machine fails to execute a task. Additionally, HDMM are very diverse systems that cannot really be classified as autonomous due to ambiguous definitions for levels of autonomy. Even then, achieving such ambiguous "highest" level of autonomy is not feasible with modern development methods, so there needs to be a more flexible way to increase automation using an incremental approach.

3.4. Theme: Management, organization, and business

Under "*Management, organization, and business*" are the observations synthesized from descriptions of how companies and business activities are organized and tools used to manage requirements, specifications, and allocation of software development tasks. Also, general notions discussing the industry and the business, are included here.

There is overlap between the themes "*Management, organization and business*" and "*Ongoing industry digital transformation*" (3.5), since they both consider the whole industry. However, "*Ongoing industry digital transformation*" warrants its own theme due to its importance and prevalence.

Main observations:

HDMM software development usually is not a singular project, but a large number of different projects that are developed with a large-scale agile process with the issue tracker as the central hub. The issue tracker is an organizational software tool to create and distribute software engineering tasks. It is the most important, but not the only communication channel or management tool. Their usage in HDMM software development is highly similar to any other large-scale software development. When projects and organizations grow enough, the tickets in the issue tracker require extensive management and prioritization.

The software always serves to fulfill some customers' wishes or requirements. However, the software developers do not face customers directly, and they are not necessarily directly aware of what the original wish or requirement was. Requirements elicitation is often done via customer interfacing or sales groups. These customer interfacing groups or managers form the requirements based on customer feedback and then send them further inside the company providing the HDMM automation software. Many problems related to HDMM software development are encountered in these requirements elicitation and specification activities. Ambiguous and undetailed requirements waste time, thus, focusing more effort on requirements and specifications would help to avoid spending time unnecessarily.

3.5. Theme: Ongoing industry digital transformation

Digital transformation is not easily defined [27], but in the context of this article, it covers quotations that describe the HDMM industry in general and business changes and problems brought by using digital technology onboard a HDMM. Digital transformation is said to encompass process digitization too, but here, we mostly focus on the *digital innovation* aspects as briefly mentioned in Berghaus and Back [28].

There is overlap between the themes "*Management, organization and business*" and "*Ongoing industry digital transformation*" (3.5), since they both consider the whole industry. However, "*Ongoing industry digital transformation*" warrants its own theme due to its importance and pervasiveness.

More specifically, "*Ongoing industry digital transformation*" includes the observations from quotations that elaborate on how the introduction of digital electronics and software impacts the HDMM industry.

Main observations:

Software's role in the HDMM industry is increasing, and its emergence has been so fast, that the industry has been unable to maintain the same pace. Traditionally, OEMs originated as machine shops, which means that their leadership has mostly understood technical matters only related to mechanical engineering and manufacturing. If such people with no knowledge or experience in digital technology decide to sell autonomous machines, the behavior of which is highly dependent on electronics and software, they will not be able to lead and manage the development of such machines. The situation is improving slowly, as the leaders and managers are being retrained or replaced, but the problem is not solved just yet. The following quotation from the dataset gives a perspective of this general issue: "*The customers in many cases, they have no idea what is even possible in that field. They run their machines today human operators and most of them cannot even imagine that it is ever possible that a machine like this could work on its own.*".

In addition to the digital immaturity of leadership, the industry's need for software engineers has risen dramatically. As the machine shops have begun creating robots, the machine design done traditionally has morphed into software development and production. When implementing intelligent machine features with software, many old and recognized problems in software engineering and computer science are encountered, for which many solutions already exist. The fields of computer science and software engineering are manifesting themselves within the machine shops, which now rediscover old problems already solved earlier in these fields.

3.6. Theme: Programming tools and model-based engineering

In "*Programming tools and model-based engineering*" the observations are built on quotations describing design-time tools used in software development. This mostly includes programming languages, IDE:s, and model-driven toolsets.

There is an overlap with "*Runtime software functionalities, structure, and underlying hardware*" (3.8) due to some ambiguity of whether some technological concepts are run-time or design-time.

Main observations:

The vast majority of software for onboard PC and offboard software is written in C, C++, or C#. Common IDE:s, such as Visual Studio are used. Some consider the C-family of languages to be outdated, especially C++ is seen as more of a hindrance to software development than an advantage. The imperative programming model of C-family languages forces the developer to pay extra care just to avoid bugs, especially so when dealing with concurrency and memory management.

The electronic control unit (ECU)-level (3) software is implemented mostly with IEC 61131-3 languages or with models. Those who use models, prefer them for two reasons: it eases

communication between domain experts and software engineers and it is more intuitive for non-software engineers to use. This intuitiveness helps especially to make sure that the domain-specific requirements for the software are met, which can be laborious with hand-written code. Models can also be misused, poorly built models are difficult to work with. The software is often made into modules that can be combined with build tools into software for a single ECU or be distributed over multiple ECU:s.

3.7. Releases and configuration

The theme "*Releases and configuration*" entails observations from quotations on how software is released and deployed to customers and how they might be configured. This theme has some overlap with "*Programming tools and model-based engineering*" (3.6), due to some ambiguity between the configuration management happening in design-time or at runtime.

Main observations:

Many companies release software in regular intervals, 2–4 per year, but some prefer or aim at rolling release. The release adoption is also up to the client, and sometimes clients do not care to update software unless there is something wrong with it. The HDMMs are used in industrial operations, and updating would require shutting the production or activity down for the update. This means the precise time for updating has to be planned ahead to avoid operational delays.

3.8. Theme: Runtime software functionalities, structure, and underlying hardware

In the theme "*Runtime software functionalities, structure, and underlying hardware*" are observations from quotations describing both the structure and intended functionality of runtime software and the hardware necessary to run the software, such as ECU:s, Programmable logic controllers (PLC:s), system-on-chips (SoC:s), sensors and actuators.

There is some overlap with "*Safety*" (3.9) and "*Programming tools and model-based engineering*" (3.6), due to implementation aspects of functional safety and ambiguity of some technologies considered runtime or design-time.

Some detailed observations in this subsection are not listed here but are collected to Table 3 shown later.

Main observations:

There is a general distinction between onboard and offboard software. Offboard software runtime environment is similar to that of information systems, i.e. servers, and the onboard runtime environment can be divided further into two levels: ECU-level and PC-level.

ECU-level runtime environment means one or multiple ECU:s that are connected together via Controller Area Network (CAN) bus. The software running in these ECU:s usually handle tasks such as power transmission, drive-by-wire, or motion control. They are embedded runtime systems that are programmed with model-based embedded system toolsets or customized IEC 61131-3 based programming tools. The ECU:s are microcontroller-level computer hardware and rarely have sophisticated operating systems. Different functionalities are sometimes separated into multiple ECU:s. Especially functional safety (3.9) is often separated into its own ECU.

Onboard PC level runtime environment means computer hardware roughly equal to PC:s in computational capability in which Linux is very often used as the operating system. It handles many functions for both autonomous and non-autonomous HDMM, but for autonomous operation, the onboard software is centered around processing and using exteroceptive sensor data to be used in motion planning. In autonomous operation, the motion planning is done at this PC level and then general motion commands are sent to the ECU, which processes them into machine-specific command signals sent to actuators.

Offboard software runs on normal IT hardware with much less restriction in computational capability when compared to onboard runtime environments. It can handle a wide variety of tasks, such as onboard software deployment and configuration management, but for autonomous operations, especially important is autonomous HDMM fleet control and operation. The offboard software has many connections to other systems, such as user interfaces, databases, or operational industry site management software. The offboard software can be much more complex than onboard software and resembles large information systems in this regard. It is connected wirelessly to the HDMM.

3.9. Theme: Safety

Under the theme "*Safety*", are the observations gained from quotations on functional safety and the effects it has on software development. There is overlap with multiple other themes, but the quotations were prioritized to favor "*Safety*" in almost all conflicting cases.

Main observations:

Guaranteeing the safe operation of autonomous machines is crucially important for their wider industry adoption. However, developing and safety certifying software is expensive and requires heavy processes. Safety certification is expensive since all system components in a safety critical system need to be certified for safety and software systems have many components or modules. These include at least the operating system, device drivers, middleware, libraries, and application logic. Because of the expenses, the technical designs for onboard systems aim to minimize the number of components, both hardware and software, that are necessary for guaranteeing functional safety. For example, one interviewee expressed the issue in this fashion: "*...we try to minimize the number of these components that have to be called as safety components, precisely because making them is unreasonably difficult.*"

In addition to minimization, autonomous HDMM safety systems are also strictly separated from other onboard PC- and ECU-level software. However, some suspect that this separation might not be enough eventually, and at some point, all onboard software has to be considered safety critical. There is a large difference in engineering culture in general software development and safety-critical systems development, so they might need to be merged at some point.

3.10. Theme: Software design and codebase evolution

The theme "*Software design and codebase evolution*" includes the observations from quotes describing how or why a codebase or a software product or project evolves over time and how they or their changes are designed. Here the codebase, software project, or software product is considered to be a central artifact that is worked on and modified to meet some goals. This modification work includes both the design or planning and the implementation or coding. The codebase's internal architecture and structure are not considered here.

Because the theme includes notions on development work, it has some overlap with "*Management, organization and business*" (3.4). The focus here is more specifically on the software design and development work.

Main observations:

Software products start from simple projects for a single customer, that are worked on until at some point they might be

labeled as products. This product then can be marketed beyond this first customer. The software is not designed or planned much initially, because often there is a need to quickly create something functional and deploy the software to the customer. Eventually, there is a system that is composed of multiple software parts and it is worked on continuously by multiple people over time. This work includes bug fixes, refactoring, or new features. There is not also a single point of when software is considered ready. It can be very simple at first and features are added over the years and updates can be developed many years after the first deployment, without really formally finishing. The codebase keeps growing and does not really decrease in size, but the development effort on certain parts can decrease.

3.11. Theme: System integration and lack of standards

The "*System integration and lack of standards*" includes the observations built from quotations on how HDMM system integration and related lack of standards impact software development. By system integration, we mean attaching and connecting different sensors, PC-level computing units, and software modules with computer networks and software interfaces. This theme has some overlap with "*Management, organization, and business*" (3.4) due to cooperative nature of such actions. The focus here is on the descriptions of tasks and goals of system integration, rather than the mechanisms or general processes described in "*Management, organization, and business*" (3.4).

Main observations:

Early integration of subsystems works better than developing the subsystems separately and then integrating them at the end of a development project. Implementing, testing and integration should be done for only one or two functionalities at a time. When integrating subsystems co-operatively, there is a high risk of failure as the integration effort is like a chain. When one party fails, the whole system will not work.

Integration is also difficult because there is a lack of widely used standards for especially onboard PC-level software. Many existing solutions are developed as in-house systems or completely embedded systems. This increases the time necessary to learn how it works and how to develop software for it. Much time is spent on simply embedding a separately developed solution into an in-house framework.

3.12. Theme: Simulation and testing

Observations on "*Simulation and testing*" are built from quotations on how software is tested and what role simulation has in the development of mobile machine automation.

Main observations:

Systematically assuring software quality is a lot of work, but it is unavoidable. If writing documentation and test cases are not done, much more time would be used in clearing confusion incurred by the bugs and the low-quality documentation. For testing, using real machines provides the most authentic data, but using them does not scale as well as they pose additional needs for time, space, and additional equipment. This makes the physical test platform easily a bottleneck and simulators help to avoid this. Hardware often also needs debugging, which needs to be done for a prototyping machine first before the software development and debugging can start.

Simulator-based testing offers many benefits when compared with physical machine testing. It scales better, it decreases the development feedback loop, simulators are easier to maintain and they offer better test repeatability. One interviewee noted, that: "*... with a simulator even if something goes horribly wrong and the machine crashes into a wall, just as an example, nothing happens. So, you just reset the simulation, find your bug, do it again. So, it gives you agility, it gives you repeatability, it gives you safety, and the last factor is automation. So, we automate the machine but we also need to automate the testing*". Simulating also helps with integration testing. Even if the data recorded from a simulator are low quality and inauthentic, they can be replaced later with better recordings as long as everything is in the same format. There are still some things that cannot be tested on simulators, but most of the bugs can be found using them. Simulators can also be used beyond just behavioral testing, such as design drafting.

There are some downsides to simulators as well. If simulated environments differ too much from real sites, it can pose problems. Also, the simulators do not transfer well between operational industries or tasks and the simulation quality might be lacking. It is difficult to balance between sufficient simulation fidelity and excessive computational load, especially for HDMM manipulation tasks, but less so for driving and navigating. There are many commercial products available for simulating various HDMM operations. However, these commercial products might be too costly, many customers purchasing HDMM software are not willing to pay much for using simulators in development.

4. Discussion

In this section, we discuss software development for HDMM automation in relation to our research questions. The answer to RQ1 will be based on our own observations. To answer RQ2, we will relate some of our findings with those found in the literature. The discussion on RQ3 leads us to a list of problem topics, which will be elaborated on further.

4.1. Forming a view of software development for HDMM and their automation

Answering RQ1: How and what kind of software is produced in the industry for HDMM automation?

From the observations, we can build an abstract view of how HDMM automation software is produced. A visual representation of the HDMM software development is shown in Fig. 6. In it, the focus is on feature requests, tickets, and the codebase. The feature requests take the form of tickets in the issue tracking system, which are then transformed into changes in the codebase by the software developers. The features originate from the client, and they are collected by a customer-facing person or group, who then initiates tickets in the issue tracker. Tickets can also be issued to include reported bugs to be fixed and codebase restructuring efforts.

The tickets will then be prioritized, modified, and split by management until something is deemed feasible to be assigned directly to the software development team. The software developers then iteratively develop and test codebase changes with design time tools, simulators, and real HDMM. Suitable codebase version branches are built at regular intervals to create products to be deployed into the operational site and their HDMM.

As for the HDMM automation runtime software, there are three different runtime environments for it. Onboard ECU is for low-level machine control, onboard PC mainly for exteroceptive sensing and motion planning, and offboard system for anything else that is not strictly necessary to run onboard. Table 3 shows more details.

To showcase the different runtime environments, example systems are shown in Figs. 7–9. Fig. 8 shows an example runtime system for both onboard PC and onboard ECU level software. This system is from an experimental HDMM (an autonomous wheel loader) built at Tampere University which is shown in Fig. 7. Fig. 9 from Bahnes et al. [29] is an example of an offboard system. It

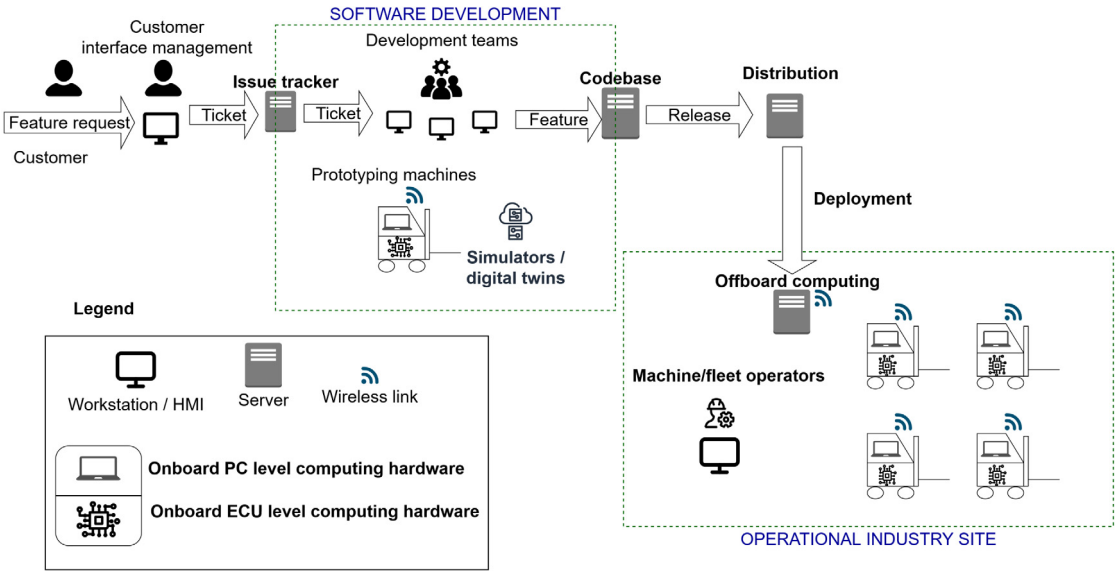


Fig. 6. Abstract view of software and its development in mobile machine automation based on the interview data. It is not an exact depiction of any actual case in particular, but rather a fusion of different instances.

Table 3
Three categories of HDMM runtime environments and the software in them.

Device category	Typical functions	Typical I/O	Typical programming languages and tools	Memory and permanent storage capacity	Typical chip type
Onboard ECU/ microcontroller	Power transmission sensing and control (such as hydraulic pressures and valves), CAN-bus interfacing, functional safety	CAN-bus, PWM, analog	C, IEC 61131-3, model-based code generation, custom tools	RAM: few MiB Permanent storage: few MiB	32-bit MCU
Onboard PC	3-D sensing, motion planning and control, machinery diagnostics, machine operator interfacing	CAN-bus, USB, ethernet, 5G or WLAN, Camera Serial Interface	C++, custom tools	RAM: Tens of MiB to few GiB Permanent storage: Hundreds of MiB to tens of GiB	32- or 64-bit ARM SoC, x86, x86_64
Offboard	Fleet control, configuration management, ERP and other management system integrations	Internet protocol stack	C++, .NET	RAM: Up to tens of GiB Permanent storage: Up to multiple TiB	x86_64

shows the operational domain of an automated container terminal. In Fig. 9, the offboard software would run in the block labeled as CBS (Confined Base Stations)

The directly mentioned tools and processes in the interviews are shown in Table 4. Please notice, that the list in the table is not exhaustive and is intended to give some idea of what technologies play a role in the HDMM automation industry.

Answering RQ2: How are HDMM software and its production different than those from general IT?

Automated HDMM are cyber–physical systems, but this characterization alone is insufficient to answer RQ2 due to the ambiguity of the definition of a cyber–physical system and the lack of established literature on the differences. Although the study made by Al-Jaroodi et al. [4] lacks empirical verification, it provides some practical framework to begin understanding the answer to RQ2. In fact, they provide one CPS case of autonomous vehicles, a category in which the HDMM fits well. The observations of our study in this article confirm a subset of their list of 10 differences, but all of them could be concerns in the HDMM software in general as well. We also incidentally confirm many findings of García et al. [3], a study which is solidly based in empirical evidence. Although our exploratory study produces more hypothetical results, we propose the following answer.

The HDMM automation software and its production are different because of their cyber–physical nature. This means the software should be real-time, fault-tolerant, and distributed but also well-integrated and it should not pose a physical risk in operation. The production of this kind of software requires different kinds of development tools for each of the three types of

Table 4
Mentions of tools, technologies and processes in the interviews in no particular order.

Mentioned tools, products, technologies and standards
PLCL, Linux, Windows, C, C++, C++17, Haskell, Rust, C#, F#, MATLAB, Simulink CodeSys, Visual Studio Code, Notepad++, Python, JSON, XML, WLAN, Ethernet, CAN bus, J1939, 4G, 5G, AUTOSAR, Qt, ROS, ROS2, MQTT, .NET, DDS, GPU, FPGA, x86, ARM Pytorch, Tensorflow, Unity, Docker, Raspberry Pi, Gazebo, git, Gitlab, Bitbucket Visual studio Polarion, Jira, Confluence, Azure, Phabricator, Jenkins, Robot Framework, E3.series Microsoft Business Central, Microsoft Dynamics Microsoft Office, E-con configurator Artifactory, Conan, Youtrack, Notion Scrum, SAFe, Kanban, Program Increment ISO 13849

Fig. 7. An example autonomous HDMM (wheel loader) that was built at Tampere University with the cooperation of industry.

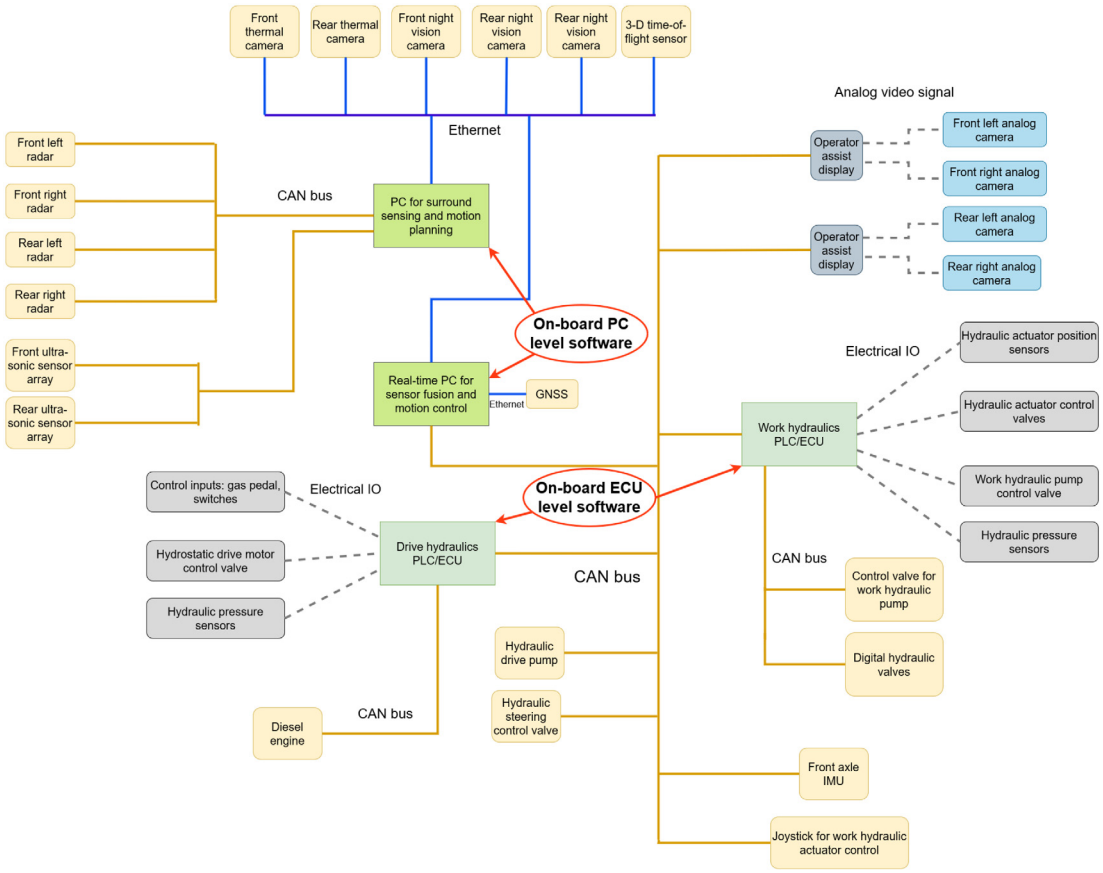


Fig. 8. Sketch of computer and electronics architecture for an autonomous HDMM from Tampere University shown in Fig. 7. It shows the different responsibilities for onboard PC and onboard ECU level runtime environment software.

runtime environments, onboard ECU, onboard PC, and offboard. Software development also requires the operational domain, mechatronics- and robotics-specific human capital, extensive testing processes and equipment for both simulated and physical settings.

Answering RQ3: What are the problems or limiting factors faced in the industry when developing HDMM software?

The answer to this is synthesized both from the observations given previously and the observation fragments given in Appendix B. Ultimately, there are many problems, varying in

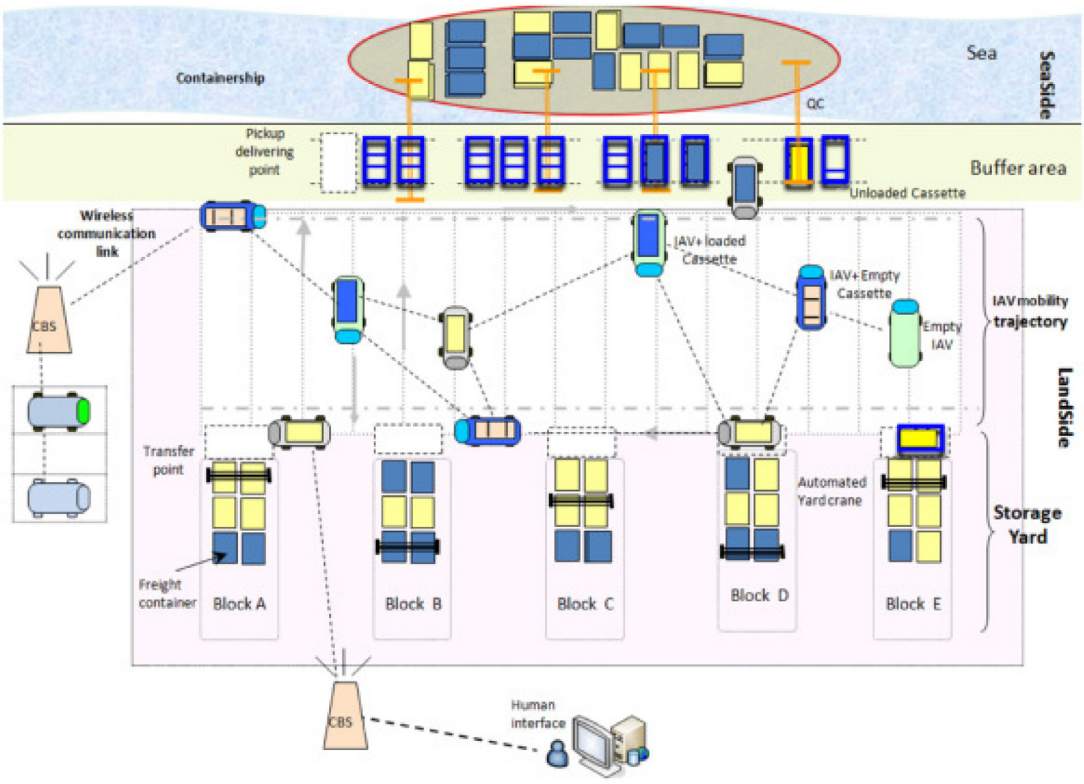


Fig. 9. Example operational domain and an offboard system application from Bahnes et al. [29] (Picture licensed under CCBY-NC-ND4.0, and it is attributed to the authors of Bahnes et al. [29] who own the copyright to it.)

specificity. Some of them are widescale problems and some are more specific bottlenecks. We focus here only on the most general problems or problem topics: *digital transformation of the industry* (4.2.1), *human capital and workforce* (4.2.2), *HDMM software standardization* (4.2.3), *simulation and testing* (4.2.4) and *safety in autonomous HDMM* (4.2.5). These general problems are discussed further in the later subsections. The discussion for each named problem topic includes a short introduction to the topic, mentions of relevant literature, and recommendations for further HDMM industry-specific research. Only a superficial treatment of these topics is possible here, as handling any of them in detail could very well be a complete study on their own. As this is an exploratory study, further work is required to fully understand each problem topic and the solutions for it.

4.2. Discussion of found problems

Here we discuss the major problems identified for the *RQ3* and connect them to the wider body of research, which helps to start the work to solve them. There are multiple problems to choose from, but here we chose the most general ones, those which solutions will have the most impact on the industry. The named problem serve also as keywords for search engine queries and the literature references serve as starter sets for snowballing [30]. This enables further exploration of the literature on the problems found in HDMM software development.

4.2.1. Digital transformation of the industry

Digital transformation is a multidisciplinary change throughout the organization [31] where business model innovations are enabled through the use of digital technologies [32] and digitalization to create and capture more value for the firm [33], through a collaborative approach and change in organizational culture [34]. However, digital transformation as a concept may be misinterpreted in different industrial contexts [31,34]. For example, as highlighted in Table B.11, HDMM companies may think of software as an add-on product rather than a key enabling technology with new value propositions. Similarly, HDMM organizations may oversell autonomous capabilities, with consequences such as delays and poor quality as noted in Table B.8. While newer organizations can adopt new technologies and build a relatively new organizational culture quickly, incumbent organizations in the HDMM industry rely on slower processes dependent on existing legacy technology and organization culture Table B.11.

Accordingly, general education around digital technologies and software engineering is very important for the HDMM industry as a whole. Incumbent organizations may need to "unlearn" some existing practices [34], since they can suffer from path dependencies and biases, i.e., we have always done things this way [33]. Such education should target the reskilling and upskilling of the workforce to improve collaboration on software-intensive topics. These educational measures can also be supplemented by creating an exploratory business unit [32], which focuses on digital technology to gain and spread digital know-how inside a company.

Furthermore, several management tensions can stem from digital transformation. To alleviate these tensions, research literature recommends the creation of a "Chief Digital Officer (CDO)" position, who guides the organization in its digital transformation [35]. Moreover, the HDMM industry would also benefit from further research on viable business models, especially for incumbents, as well as research on improving the education and cultural transformation of the organization for more agile and interdisciplinary ways of working.

4.2.2. Workforce and human capital

Lack of workforce is a global societal issue. Moreover, the HDMM industry needs to compete against large technology companies for the workforce that possess suitable digital skills [33,36]. Unfortunately, especially the younger demographic in the available workforce prefer the tech giants, such as Meta, Amazon, Apple, Netflix, and Alphabet [33]. These broader societal and demographic trends compound the shortage of skilled software engineers and IT professionals available to the HDMM industry, thereby delaying or even stalling software development projects.

Since this issue of workforce shortage is global, addressing it requires very wide actions. Such actions include but are not limited to investments in economic, legislative, and educational reforms to develop, attract, and retain new software engineering workforce and skills that are urgently required in society [31,37]. Furthermore, it is imperative that future research takes a multidisciplinary perspective along with cooperation between academia and the industry, to holistically address the workforce shortage issues. Such multidisciplinary research could be used to inform educators, policymakers, regulators, and legislators about the future needs and wants of the HDMM industry and society.

Human capital

Building HDMM-specific human capital is mostly a matter of education, but the different disciplines required in HDMM automation are still somewhat separate. There is no single source for basic information on the topic, but here we will list starting points to start building the skill set needed to build an intelligent HDMM. The operational industries will be omitted, even though they are very important especially when the level of autonomy increases for the machines. Further discussion on this topic of intelligent HDMM education can be found in [36,38].

Software engineering itself is fairly mature, and there are multiple sources containing the central information of the discipline. One of these central works is the Software Engineering Body of Knowledge, [24]. For the robotics discipline in general, a similarly central piece is the Springer Handbook of Robotics, [39]. A source for more HDMM-related robotics subarea, mobile robotics, is [40].

The book from Lee & Seshia [25] is a comprehensive work on computing in cyber–physical systems in general. It combines the modeling aspects of both the continuous models of the physical world and the discrete models found in computer science. For the mechanical engineering aspects of the HDMM and especially their power transmission and control, a central piece is from Geimer [9].

Further work is necessary to combine all the disciplines mentioned previously. Fusing all the necessary knowledge can take many forms. For example, educational institutes can create degree programs for robotics with the possibility to specialize in HDMM automation. However it might be done, further cooperation with industry is necessary.

Developer onboarding

Onboarding means the processes necessary for a new hire to go through in a company to become productive quickly. For software developers, one important part of this is learning the codebase. A wide body of literature relevant to this general topic can be found under terms of *onboarding* or *organizational socialization*, but not all of the discourse is specific to software developers or engineers. Very detailed work on general software engineering onboarding is presented by Yates in [41]. It contains, *inter alia*, multiple recommendations for improving onboarding sessions. An empirical study in an industrial context is portrayed by Ju et al. in [42], where they address developer onboarding tasks and strategies. Their evidence led to three onboarding strategies, Simple-Complex, Priority-First, and Exploration-Based, of which the Simple-Complex was deemed the best for junior developers.

4.2.3. Software standardization

Standardization of complex systems, such as automated HDMM, evolves with the enabling technology and the markets will ultimately define the standards [43]. Here we will merely point to promising directions for standardization instead of ruling single technology or a standard as a solution. The HDMM:s role in operational industries complicate software standardization, as the operational industries and their standards are very distinct those from robotics even if the HDMMs are becoming robotic.

Perhaps the best option for offboard software is to follow operational industry site management software and related standards. For example, Terminal Operating Systems [44] are heavily used in container ports, and building information models (BIM) [45] are used in construction. This offboard software could then be connected to individual HDMM through more or less industry-specific fleet control solutions.

The onboard ECU-level software-related standardization is not an issue as it has matured now for a while. The main connectivity interface is CAN bus and the programming is done directly in C or IEC 61131-3 languages or code is generated from models. But for onboard PC software, the situation is much more complicated now because there are so many IoT standards to choose from Vivek et al. [46]. There are promising solutions that seem suitable for HDMM automation, such as ROS2 and DDS [47].

Because of the multitude of options for onboard PC software standards, it is possibly best to simply try them in practice. For doing this systematically and fast, it would help to create a case study template or some evaluation criteria. These could be used to evaluate each IoT standard suitability for different HDMM automation scenarios.

4.2.4. Simulation and testing

There are some differences between robotics simulation and HDMM-specific simulation. The HDMM industry will however benefit from more general robotics simulation as the automation level increases. When it comes to HDMM-specific simulation, the literature is somewhat scarce, but the Ph.D. work of Lauri Luostarinen [48] provides a good view of what the HDMM-specific simulation focuses on: powertrain modeling, hardware-in-loop testing, and virtual reality for operator training.

There are plenty of existing works on more general robotic systems simulation and testing. A good example is a survey from authors of Afzal et al. [49], which is based on both qualitative and quantitative data. They state, that the simulation fidelity issue is well understood by the robotics community, and point to creating useful models to be a perhaps more important topic to focus on in the future. They also recommend that further software engineering work is required for domain-specific languages for scenario construction, simulator reliability, and automation of simulator-based testing. Another good example is from authors of Choi et al. [50], where they point out problems and propose solutions to address the problems. These solutions include: improving open-source platforms (especially for soft robotics), more work on modeling robots, their subsystems and human–robot interactions, creating simulation competitions, and utilizing the latest computer hardware in the simulators. Also, the Ph.D. thesis of Afsoon Afzal [51] provides more details on the topic of simulation-based testing for robotics. From our point of view, perhaps the most important contribution from Afzal [51] is that the author argues for low-fidelity simulation being a cost-effective way to find many bugs in robotic software.

If we combine our findings, and those from Afzal [51], Afzal et al. [49] and Choi et al. [50], it would be fair to say that there definitely is a need for further development of simulation models. There is no single simulator that could simultaneously handle all the aspects that need to be tested. These aspects can be different, like those in [50], where the authors mention five different

parts of robotics: robots, sensors, synthetic worlds, humans, and communication layer. We suggest a different scheme, building from the ideas in [50]. Our division for different layers of modeling and simulation would be around the runtime environments: powertrain and hardware-in-loop simulation for the runtime ECU software, synthetic worlds and sensor simulations for operator training and testing onboard PC software, and a separate layer for fleet-level simulation for testing offboard software.

4.2.5. Safety in autonomous mobile machines

Highly automated HDMM application areas exceed the scope of guidelines given in existing machine safety standards, which reveals limitations in machine safety conformance. Similar issues emerge in other industries, such as when the level of automation increases in road vehicles. This deficiency in the current standards for road vehicles prompted an analysis [52] of automotive standard ISO 26262 [53] and functional safety. This analysis inspired the development of ISO 21448:2022 [54]. ISO 21448:2022 supplements ISO 26262 by providing risk mitigation efforts related to the safety of the intended function (SOTIF). Safety of the intended function is defined as the absence of unreasonable risk due to hazards resulting from functional insufficiencies of the intended functionality or its implementation [54]. Increasing the level of automation in HDMM reveals similar limitations (like those in road vehicles) with the definition of functional safety in existing machine safety standards, which limits the safety conformance of the highly automated HDMM.

Within the robotics domain, the authors of Salvini et al. [55] analyze robotic standard ISO 13482:2014 [56] for personal care robots. They explain, how the hazard definitions underlying the safety requirements are insufficient. The increasing automation of personal care robots introduces new hazards that the existing standard does not account for.

Similar emerging limitations of standards, in domains related to HDMM, inspire the development and publication of machine safety-specific standards for different HDMM operational domains. For example, ISO 17757 for highly automated and autonomous earth moving and mining machines [57], ISO 18497, Agricultural machinery, and tractors — Safety of highly automated agricultural machines [58], ISO 3691-4, for driverless industrial trucks guiding design for unmanned forklifts, automated guided vehicles and other related machines [59]. However, none of them discuss the supplementation of an HDMM-specific definition of functional safety with a definition of SOTIF, and they still rely on a human operator who is ultimately responsible for safety. This reveals that safety conformance for highly automated HDMM requires further research.

5. Threats to validity

This study is an exploratory study with qualitative analysis. As described in Wohlin et al. [20], we identify four qualitative analysis specific aspects of threats to the validity: construct, external, internal and reliability.

5.1. Construct validity

The construct validity focuses on how researchers' intent might differ from what was actually researched. The main threat to construct validity comes from the cross-disciplinary nature of HDMM automation and the very broad topic of software engineering. The interview participants had multiple educational backgrounds, as shown in Table 1. This can make the terminology used in the interviews ambiguous and overloaded, which threatens the validity of the used constructs.

Another threat to construct validity comes from the similarities and differences between robotics software engineering and HDMM automation software engineering. We stated earlier, that they are separate cases, even if there are many similarities. However, these differences and similarities are not yet documented in the literature. Therefore, future studies are highly necessary for establishing the difference in software engineering between mobile and industrial robotics and HDMM automation in order to address this threat to construct validity.

5.2. External validity

External validity is concerned with generalizing the findings. This study did not perform formal a quantified study with a statistically determined sample size and a random sampling process. This can result in selection bias, which in turn reduces external validity. Also, the interviewees were all from Europe and this geographical specificity is a threat to external validity as there is no guarantee that HDMM industries and the technology elsewhere in the world would be exactly similar.

5.3. Internal validity

Internal validity looks at causal relationships between constructs. As this study was not focused on finding causal relationships, there is not much threat to its validity either. The sheer complexity of the topic can pose some threat to internal validity.

5.4. Reliability

Reliability is threatened when the results are dependent on particular researchers. This study's reliance on qualitative analysis is a threat to reliability, as there is no triangulation with other sources of data. Also, the exploratory nature and the resulting ambiguity in the interview questions are a threat to reliability.

6. Conclusions and future work

In this study, we explored the topic of software engineering in heavy-duty mobile machine automation based on interviews of professionals working in this industry. The main attainments of this exploration were: an overview of what kind of software there is to automate HDMM, how it is developed, and what problems are faced in its development. As an exploratory study, our results help to form research directions for further industrially relevant and more detailed studies.

We present the following characterization of HDMM automation software based on the analysis of the interviews: The automated HDMM are cyber–physical systems embedded into wider production systems in the operational industry. This automation necessitates knowledge of various fields, such as software engineering, robotics, mechatronics, and the operational industry. The runtime software for automated HDMM is found in three layers, offboard, onboard PC, and onboard ECU. The offboard software handles computation that is not necessary to run onboard, such as fleet control. The onboard software handles robotics-specific functionality such as actuator control at the ECU level and motion planning and exteroceptive sensing at the PC level.

The problems the HDMM industry faces with regard to software engineering, when automating the machines, are broad and not necessarily directly solved by any specific technology. Future work could help the HDMM industry by studying further both technical and non-technical problem areas. Relevant, less technical future research directions could be: advancing the digital transformation of the HDMM industry and investigating and addressing the workforce issues in more detail. The latter topic is highly correlated with education, and thus perhaps more work is

warranted on improving the education of HDMM automation in order to build more relevant human capital and available workforce. The HDMM industry would benefit from more technical research on how to guarantee safety for autonomous machines and how and what simulators to use and to determine their role in software testing. The industry would also benefit from further studies on finding the most suitable standards that help to cooperate and manage cooperation and design complexity.

Finally, it would benefit both industrial practitioners and researchers if the differences and similarities between robotics software engineering and HDMM automation software engineering were studied and documented. The two share a great deal, yet are different in some aspects. The industry could then perhaps use this knowledge to decide between reaching out to the robotics community for solutions when the contexts are similar enough and addressing independently the more industry-specific problems.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Andrei Ahonen reports financial support was provided by FIMA Forum for Intelligent Machines ry. Aimee de Koning reports a relationship with EPEC Oy that includes: funding grants. Tyrone Machado reports a relationship with Bosch Rexroth AG that includes: employment.

Data availability

The data that has been used is confidential.

Acknowledgments

This work was financially supported by the Doctoral School of Industry Innovations at Tampere University, Finland and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 858101.

Appendix A. Interview and qualitative analysis methodology

A.1. Qualitative analysis

Thematic analysis [21] with ATLAS.ti 9 software tool was used to analyze the transcribed data. The coding was done by the first author and the fourth and fifth authors worked as validators. Even though the validators were two different persons, at inter-rater agreement level calculations they were treated as a single validator. The term *code* and *theme* are considered equivalent for purposes of this appendix and the article it is part of.

Transcription work and initial coding
The voice recordings of the interviews were transcribed by the first author, after which the interview data was anonymized by removing names and details pointing out to a single person or a company. After the transcription, the first author read through whole dataset multiple times. After this reading, an apriori list of codes/themes was formed and used in initial phases of the coding. Rework of this coding schema was done multiple times before proceeding to validate the schema. During this initial coding before validation there were varying levels of hierarchy between the codes, and the coding was not mutually exclusive. However, the final code scheme was made mutually exclusive for easing the validation process, but some semantic overlapping was unavoidable. The overlaps are mentioned in the theme descriptions in the article. All codes were in English even if the language used in the interview itself was not. All participants of validation knew Finnish and English.

First validation
There were 421 quotations under 26 codes when the first validation was begun. Before proceeding with validation itself, a goal for agreement rates was discussed among the first, fourth and fifth authors. A goal was set upon Cohen's kappa value 0.6 representing moderate agreement [60] due to the cross disciplinarity of the topic. After setting this goal, a spreadsheet with a list of all the quotation ID:s from the qualitative data analysis tools was created. This spreadsheet also contained functionality to take a number of random samples without replacement from this list of quotation ID:s. Also a simple document was made by first author, that explained and described the codes under validation. This document, the spreadsheet and the software tool project file was sent then to the validators. In this software tool project file, the quotations were free and the validators had no knowledge of how each quote was actually coded by the first author.

To reach a $\sim$ 10% representative set of random samples, both validators took 20 random samples using the spreadsheet function. There was a random chance for validators to pick same quotations with the spreadsheet tool, and this happened for one quotation. However, when the validators proceeded to code the sampled free quotations, the inter-rater agreement rate was very low. Even though the kappa value was set upon 0.6, it was not necessary to calculate it because the simple agreement rate was $\sim$ 0.4. Because of this low agreement rate, a rework of the coding schema was done.

Second validation
After the rework, similar validation setup was done with 373 quotations under 12 codes. After validators had coded the randomly sampled free quotations, the initial value for the agreement was much higher than in previous, $\sim$ 0.8. Before proceeding to calculate final kappa value, a discussion was conducted for the conflicting quotations. For the conflicting quotation, the initial coder provided the original coding of that quotation a rationale for it. Then the validators could either agree or disagree with it and possibly change their coding based on this rationale. This way the agreement was reached on all codes except 1, which was then split into multiple quotations each under a code everyone agreed upon. This coding scheme is the set of themes presented in the article.

A.2. Interview invitation

The following is the general invitation that was sent over email to the potential interviewees. The email also had three attachments, the interview guide which was meant to steer the discussion, information sheet of the study itself and the privacy notice for GDPR both for following ethical research guidelines.

"*Hello,*

I would like to invite one or more of your company's employees to be interviewed for a study: Survey of software engineering in mobile machine automation

As the study name implies, the goal of this research is to map the current state of software engineering used for automating mobile machines and the problems encountered. The ideal interviewee would have a great deal of experience in developing software for mobile machines.

The interview will last 1 h at most. It will be conducted over network with Zoom-teleconferencing application and it will be recorded and transcribed for analysis. The interview does not have fully predefined list of questions, but there is an interview guide document, which is attached as "Haastatteluohje.pdf".

Table A.5
Interview transcription details.

Transcript ID	Person ID	Transcript length (pages + non-space characters)	Coded quotations
2605	1	15 pages, 38 234 characters	26
0705 1205	(2 & 13) + (3 & 13)[a]	48 pages, 93 770 characters	81
0706	4	14 pages, 32 210 characters	25
0806	5	12 pages, 24 175 characters	14
1005	6	12 pages, 30 340 characters	24
2406b	7	10 pages, 24 816 characters	17
1006	8	12 pages, 25 423 characters	19
1405	9	9 pages, 24 744 characters	24
1406	10	15 pages, 34 876 characters	24
2005	11	12 pages, 23 566 characters	26
2806	12	13 pages, 29 549 characters	20
2105	14	12 pages, 25 171 characters	22
2306	15	17 pages, 45 521 characters	34
2406a	16	12 pages, 34 770 characters	17

[a]Merged transcript of two interviews, each with 2 interviewees with person 13 present in both.

If you accept this invitation, I would kindly ask you to suggest a suitable time for the 1 h interview. When we have reached an agreement on the date and time, I will send an Outlook calendar invite for a Zoom meeting.

With this invitation, there should be an information sheet, research privacy notice document and the interview guide ("Haastatteluohje.pdf").

I can answer any questions via email or by a phone at any time before the interview. Questions can also be asked via Zoom immediately before starting the recording for the interview."

A.3. Interview guide

The following is the English language part of the interview guide document, which was sent to the interviewees.

"*This document serves as an interview guide for the study "Survey of software engineering in mobile machine automation". The interview is roughly sectioned into 3 themes: overview, domain specificity and challenges. In the following part, the themes are listed with the subtopics that the questions will be based on. The interview is not meant to follow this template exactly, but interviewees can steer the discussion to directions they see relevant.*

Theme 1: Overview of current state of the industry

- *Interviewee POV: background, years of experience both in software and mobile machinery domains*
- *Overview of software lifecycle: requirements, plan/design, implementation, deployment, monitoring, maintenance*
- *Tools, methodologies, frameworks and paradigms used in software lifecycle*
- *Code reuse: internal and external, open source*[3]

Theme 2: Domain specificity

- *Typical subsystems*
- *Role of level of automation*
- *Role of simulators in testing*
- *Role of hardware: components and test systems*
- *Role of domain knowledge, areas of knowledge*
- *How might mobile machine automation be different from other software intensive domains?*

Theme 3: Challenges

- *Limiting factors and major problems*
- *What activity is time consuming and/or error prone?*
- *Wrong approaches to specific problems*
- *Successful approaches to specific problems*

"

[3] This topic was de-prioritized and was not discussed in most interviews due to lack of time.

A.4. Interview transcription detauks

See Table A.5.

Appendix B. Fragmented observations

See Tables B.6–B.17.

Table B.6
Observation fragments for "*Difference to IT*".

ECU-level (3) software is built around signals when compared to software built around data in general programming. This has led to ECU-level functionality to be easier to model, which has made it easier to adopt model based software development.
Many HDMM automation systems are acceptance test driven, which makes their deployment is much more difficult than it is for user interfaces or databases.
User interface design is not nearly as important as in general IT systems. Most of the software interacts only with other software through various communication protocols.
Data models are simple when compared to information systems, which might have complicated business rules and relationships between objects. This results in much smaller role for databases in a runtime system.
The onboard hardware has to have a track record and the runtime environment has less redundancy since you cannot replace misbehaving embedded computers like you can replace virtual machines running in data centers. This can lead to outdated computational platforms, since the hardware release cycle is long.
Debugging real-time systems is different from general information systems. Debuggers are not as useful since you cannot stop the real world. Instead, debugging relies on collecting logs and the system behavior is inferred from them. This affects testing as well.

Table B.7
Observation fragments for "*Human capital and workforce*".

Autonomous machine fleet control is quite abstract application and working on it requires mathematical modeling and algorithmic thinking. Developers need to be able to do suitable approximations and heuristics that are necessary to reduce the already massive search space in optimization tasks for fleet operation.
Solving problems in the operational industry can lead to endless emergence of corner cases if the developers lack domain knowledge.

(*continued on next page*)

Table B.7 (*continued*).

Only the biggest organizations might have enough human capital to design an entire autonomous HDMM from scratch. For smaller companies, there needs to be good business networks to reach autonomous HDMM.
Configurability of runtime software can reduce the need for domain knowledge in software development. The clients can make the necessary domain specific adjustments to the software on site. (Conceptual overlap with "*releases and configuration*")
It is beneficial to use same programming languages and tools for all developers in an organization, as this enables efficient knowledge sharing. Learning a new programming language or other major tool is a risky effort for a whole organization and takes a long time.
There are some subsystems that do not need much operational industry or HDMM knowledge for implementation, such as localization and navigation or general user interfaces.
People outside of robotics do not often understand the significance of properly done 3-D sensing calibration.
The HDMM development engineers have wider set of responsibilities and they cannot specialize as much as those in automotive industry, where an engineer can be narrow-domain expert.

Table B.8
Observation fragments for "*Management, organization and business*".

Management tools should be matched with the organization size, i.e. small agile companies should not use enterprise level management platforms, which would hinder their agility.
For some, the requirements elicitation happens during sales process, and not during the implementation.
For software specific technical problems, there are not many bottlenecks, since there already are solutions for many common problems. The specification and requirements issues pose more difficulties.
Overselling autonomous capabilities sometimes happen, which can lead to unseen delays as the promised functionalities need to be developed and tested first before they can be deployed. Excessive promises can also lead to too quick deployment, which means there is no time to develop proper testing infrastructure.
Scrum, Kanban and Scaled Agile Framework are commonly used management methods, with some organization specific adjustments often done. Agile methods are still new to some companies.
In larger companies, R&D works on internal software tooling and long term autonomous HDMM projects specific to the company, outside of immediate customer needs.
Partnerships and collaboration can be difficult to organize, if companies are geographically separated, have different processes and cultures.
Many different autonomous HDMM R&D projects with integration intensive phases share similarities which can be formed into processes.
Software and its provided capabilities might not be a marketing asset for OEM:s just yet. The traditional engineering features, such as mechanical capabilities, still weigh heavily in HDMM markets.
Software development and production is largely limited by resourcing, and not by technology in itself. Almost anything can be achieved to some degree when sufficient resources are dedicated to software development.

Table B.9
Observation fragments for "*Management, organization and business*".

GDPR might make delivering certain features impossible, even if the customer would want the feature.
Occasionally wasting time on something that does not work, is not necessarily a wrong approach, it is just the agile way of working.
the industry is at similar position with automation of HDMM like it was with 1960's with mechanical engineering. Many have their own solutions and competing products have not yet converged into similar designs.
Cooperation of multiple autonomous machines from different OEM's is still far into the future.
Compared to automotive industry, HDMM OEM:s have clear responsibility for their products and their safety. Also the HDMM products are business investments and not consumer products which shifts the product features into different directions.

Table B.9 (*continued*).

The onboard computing equipment should not be too expensive, but the details are dependent on machine type. Larger machines have much higher manufacturing and deployment costs and thus are more tolerant to increased computing equipment expenses. The onboard electronics of a HDMM will be replaced multiple times before the mechanical chassis is decommissioned.
Software R&D projects often suffer from moving goalposts. More and more complexity inducing features keep being added to a product which makes delays the project. This also risks having still a nonfunctional product at the end of the project. Changing targets between sprints also delays finishment of functional products.
Agile methods are often misunderstood to disregard planning altogether and this can lead to losing the focus of activities. The physical aspects especially in prototyping makes planning and design more useful, as changing hardware takes more time and effort. For more hardware intensive projects, waterfall-like processes are still beneficial.
Conway's law holds also for mechanical hardware.

Table B.10
Observation fragments for "*Level of autonomy*".

Autonomy is largely, but not completely, implemented with software.
A high level of autonomy for a single machine can be less important than the overall performance of a whole fleet at the operational industry site.
The role of offboard software in level of autonomy is significant. A single autonomously operating machine might not be able to reroute their path around an obstacle, because the trajectory information comes from the offboard software. And there are economical benefits to relying more on offboard software in autonomy, because decreasing the unnecessary onboard computation also decreases the necessary investments on the physical machine.
There is economical benefit in increasing level of autonomy in some manually operated machine types to make certain tasks less dependent on highly experienced operators.

Table B.11
Observation fragments for "*Ongoing industry digital transformation*".

In some operational industries, there are emerging sophisticated digital models which serve as basis for operator assistance functions in HDMM. If the HDMM operators do not understand this digital model, it means the operator assistance feature has no value.
Companies that have recently begun to automate their machines are not always aware of what is feasible to implement with software and what is not. There might be expectations of finding off-the-self solutions for highly autonomous behavior, where there are none or how design and implementation of software enabled features would cost in range of tens of thousands $/€, instead of more realistic range costing millions of $/€.
Software engineering as a discipline is young compared to the mechanical engineering where HDMM have their roots in. Mechanical engineering and machine design is much more mature, and there are hopes that software engineering would mature as a discipline which would help managing the software projects.
Some traditional machine shops might lack human capital and knowledge about digital technology to the degree that they cannot effectively purchase software products necessary for HDMM automation.
The digital immaturity shows in processes not being fit for software work or unwillingness for investing enough in the software, even if the software might be 80% new products value. Resulting under-resourcing means understaffed projects where everything is done under high time pressure.
Some companies have no established processes for software production.
Many HDMM customers in the operational industry might not even know yet what is possible and what kind of tasks a HDMM can perform without an operator using current technology.
Agile way of working can be a new thing for the people in HDMM industry, since mechanical engineering work still follows more traditional management models. Also there is a lot of trust placed in tradition and many doubt any new technology, especially when the company has been in business for a long time.

(*continued on next page*)

Table B.11 (*continued*).

Many HDMM companies still think software more as an addon to an existing product, rather than consider it as key technology and the driver of new value.
Old companies have also much existing legacy technology and culture to consider. New companies can adopt new technologies easier.

Table B.12
Observation fragments"*Programming tools and model based engineering*".

Some have had positive experiences in implementing runtime software using functional programming languages, such as Haskell or F#. They are deemed especially suitable for more mathematical and algorithmic problems. Those developers who have learned functional programming, have reported it to be a great improvement over imperative programming. Using functional language has enabled for the developers to focus on the application area rather than managing the computer for unnecessary details.
Relying on open source software can impose some problems, as there is no traditional client-customer relationships for the used development tools. One cannot simply contact the supplier and ask them to fix errors.
For producing onboard PC-level software for 3-D sensing, having properly calibrated measurement data is crucial before development can start.

Table B.13
Observation fragments for "*Releases and configuration*".

Onboard software for both PC and ECU level is preferred to be easily configurable for various IO mappings and HDMM types. Some consider configuration management being laborious in general, but others have found suitable tools. The configuration management can be handled in the ERP-level software or the configuration management can be connected to it.
Some use OS-level virtualization in deployment for onboard PC software.
When ECU-level onboard software is deployed to an HDMM, the software has to be able to be configured for various situations and IO mappings. There are various tools to achieve this, with EDA software.

Table B.14
Observation fragments for "*Runtime software functionalities, structure and underlying hardware*".

Some examples of the onboard PC-level functionalities are HMI for machine electronic system and operator assistance, localization and navigation, kinematics calculations, predictive maintenance and exteroceptive sensor data processing from cameras and lidars necessary for autonomous operation. Onboard machine learning inference systems are also run at the PC-level, often combined with hardware acceleration.
For correct functioning, it is very important to extrinsically calibrate exteroceptive sensors before they can be used by the software, but this aspect is not yet known by people that do not have much experience in robotics.
CAN-bus in HDMM is ubiquitous, but it is very slow and considered by some to be outdated and error-prone.

Table B.15
Observation fragments for "*Safety*".

It is very much unclear how undeterministic controllers developed with machine learning methods can be made functionally safe.
Simple safety system can be a proximity sensor connected to an safety PLC that overrides other ECU:s and stops the machine safely if detects objects that are too close.
For economical reasons, reusable safety systems are preferred. Also it needs to be decided early on in a product development phase if it is meant to be certified for safety and marketed as such.
Increasing the level of autonomy means also increases the need for safety.
Sometimes there is a no way to provide an autonomous machine to a client, because it would be completely infeasible to guarantee its functional safety.

Table B.15 (*continued*).

Safety standards are lagging behind for software. Hardware has clear concepts and definitions for reliability but similar notions for software are not as finished yet.
The customers of autonomous HDMM industry expects the OEM:s to take responsibility of guaranteeing safe operation of the machines. The acceptance for safety failures will be very low, especially in the beginning of their adoption.
In some cases it might not be economical to safely automate machinery or their functions. One technical solution for allowing safe autonomous operation is supervised autonomy. In it, there exists an autonomous machine or a fleet which can be stopped by an operator who constantly supervises the operation. It can make economical sense in some cases or industries and be less profitable in others.

Table B.16
Observation fragments for "*Software design and codebase evolution*".

The software evolution has to be balanced, so that it does not start to skew to favor only a few customers, displeasing others. Software product management is necessary, and configuration management can help with handling more customer specific changes.
When codebase has grown large enough, dedicating more time for design and planning before writing code is preferred. Also rework on whole architecture is often necessary before further features can be implemented.
End user needs should be considered as the starting point for all requirements and at first system is viewed implementation agnostic at early design phases. Some functions can be implemented with hardware or software, but when hardware is more involved in the implementation, more time spent in planning and designing phase is preferred. Design and implementation work should still be done in small increments, instead of creating a large system at once.
Data collection and data engineering for onboard machine learning models can take months, even years, before even any model training can be done.

Table B.17
Observation fragments for "*Simulation and testing*".

Creating simulations for testing offboard software is easier than for onboard software. Operational industry site is preferred to be simulated. There is not much need to use real machines for fleet control level software testing and other offboard tasks.
Logging systems are important to testing and debugging. Most of the diagnosis of software runtime errors is based on logs, which can be used for both testing the software while developing software and monitoring the software after deployment. For example, a customer might send logs from a machine that has behaved erroneously in production and the developers can inspect the logs and infer the reason for errors. Replaying recorded logs can aid in developing automated testing setups such as those seen in for information systems software. These automated tests are used by some and they are important as they help with checking many corner cases and with avoiding regressions.
Making changes into a real machine requires more effort and resources than with simulators and the physical features can lag behind design decisions for new machine models to be tested. For small features or testing tasks, using real machines already located in a suitable test site is preferred because it provides more realistic data and the test preparations can be done quickly.
Some companies have to create internal testing tools because lack of viable external alternatives and this takes time away from algorithm development.
The amount of corner cases because of the physical nature of the HDMM domain is so vast, no one can cover all of them with testing. Testing even a reasonable subset takes a lot of time and is costly.
Automated HIL-testing is often done for ECU level software.
There is an autonomous machine specific level of testing called work cycle testing, which is done after integration testing.
Machine learning models can be pretrained with simulated data, before they are finalized with real data.
There is a lot of work to be done for HDMM simulation in general, it is a very complex topic.

References

[1] X. Zheng, C. Julien, Verification and validation in cyber physical systems: Research challenges and a way forward, in: 2015 IEEE/ACM 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems, 2015, pp. 15–18, http://dx.doi.org/10.1109/SEsCPS.2015.11.

[2] K. Feichtinger, K. Meixner, F. Rinker, I. Koren, H. Eichelberger, T. Heinemann, J. Holtmann, M. Konersmann, J. Michael, E.-M. Neumann, J. Pfeiffer, R. Rabiser, M. Riebisch, K. Schmid, Industry voices on software engineering challenges in cyber-physical production systems engineering, in: IEEE International Conference on Emerging Technologies and Factory Automation, 2022.

[3] S. García, D. Strüber, D. Brugali, T. Berger, P. Pelliccione, Robotics software engineering: A perspective from the service robotics domain, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, http://dx.doi.org/10.1145/3368089.3409743.

[4] J. Al-Jaroodi, N. Mohamed, I. Jawhar, S. Lazarova-Molnar, Software engineering issues for cyber-physical systems, in: 2016 IEEE International Conference on Smart Computing, SMARTCOMP, 2016, pp. 1–6, http://dx.doi.org/10.1109/SMARTCOMP.2016.7501717.

[5] T. Bures, D. Weyns, C. Berger, S. Biffl, M. Daun, T. Gabor, D. Garlan, I. Gerostathopoulos, C. Julien, F. Krikava, R. Mordinyi, N. Pronios, Software engineering for smart cyber-physical systems – towards a research agenda: Report on the first international workshop on software engineering for smart CPS, SIGSOFT Softw. Eng. Notes 40 (6) (2015) 28–32, http://dx.doi.org/10.1145/2830719.2830736.

[6] T. Bures, D. Weyns, B. Schmer, E. Tovar, E. Boden, T. Gabor, I. Gerostathopoulos, P. Gupta, E. Kang, A. Knauss, P. Patel, A. Rashid, I. Ruchkin, R. Sukkerd, C. Tsigkanos, Software engineering for smart cyber-physical systems: Challenges and promising solutions, SIGSOFT Softw. Eng. Notes 42 (2) (2017) 19–24, http://dx.doi.org/10.1145/3089649.3089656.

[7] T. Bures, D. Weyns, B. Schmer, J. Fitzgerald, Software engineering for smart cyber-physical systems: Models, system-environment boundary, and social aspects, SIGSOFT Softw. Eng. Notes 43 (4) (2018) 42–44, http://dx.doi.org/10.1145/3282517.3302401.

[8] T. Bures, D. Weyns, B. Schmerl, J. Fitzgerald, A. Aniculaesei, C. Berger, J. Cambeiro, J. Carlson, S.A. Chowdhury, M. Daun, N. Li, M. Markthaler, C. Menghi, B. Penzenstadler, A. Pettit, R. Pettit, L. Sabatucci, C. Tranoris, H. Vangheluwe, S. Voss, E. Zavala, Software engineering for smart cyber-physical systems (sescps 2018) - workshop report, SIGSOFT Softw. Eng. Notes 44 (4) (2019) 11–13, http://dx.doi.org/10.1145/3364452.3364465.

[9] M. Geimer, Mobile Working Machines, SAE International, Warrendale, PA, 2020.

[10] ISO, Robotics — vocabulary iso 8373:2021, International Organization for Standardization, 2016, URL https://www.iso.org/standard/75539.html.

[11] T. Machado, A. Ahonen, R. Ghabcheloo, Towards a standard taxonomy for levels of automation in heavy-duty mobile machinery, in: ASME/BATH 2021 Symposium on Fluid Power and Motion Control, 2021, http://dx.doi.org/10.1115/FPMC2021-70251.

[12] W.R. Haycraft, History of construction equipment, J. Construct. Eng. Manage. 137 (10) (2011) http://dx.doi.org/10.1061/(ASCE)CO.1943-7862.0000374.

[13] ISO, Standards catalogue for agricultural machines, URL https://www.iso.org/ics/65.060/x/.

[14] ISO, Standards catalogue for construction equipment, URL https://www.iso.org/ics/91.220/x/.

[15] ISO, Standards catalogue for materials handling equipment, URL https://www.iso.org/ics/53/x/.

[16] ISO, Standards catalogue for mining equipment, URL https://www.iso.org/ics/73.100/x/.

[17] ISO, Standards catalogue for robotics, URL https://www.iso.org/ics/25.040.30/x/.

[18] D. Fassbender, V. Zakharov, T. Minav, Utilization of electric prime movers in hydraulic heavy-duty-mobile-machine implement systems, Autom. Constr. 132 (2021) 103964, http://dx.doi.org/10.1016/j.autcon.2021.103964.

[19] A. Haghighatkhah, A. Banijamali, O.-P. Pakanen, M. Oivo, P. Kuvaja, Automotive software engineering: A systematic mapping study, J. Syst. Softw. 128 (2017) 25–55, http://dx.doi.org/10.1016/j.jss.2017.03.005.

[20] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, first ed., Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, http://dx.doi.org/10.1007/978-3-642-29044-2.

[21] V. Braun, V. Clarke, Using thematic analysis in psychology, Qual. Res. Psychol. 3 (2) (2006) 77–101, http://dx.doi.org/10.1191/1478088706qp063oa.

[22] P. Eriksson, A. Kovalainen, Qualitative methods in business research, london, 2008, http://dx.doi.org/10.4135/9780857028044.

[23] C. Wohlin, D. Šmite, N.B. Moe, A general theory of software engineering: Balancing human, social and organizational capitals, J. Syst. Softw. 109 (2015) 229–242, http://dx.doi.org/10.1016/j.jss.2015.08.009.

[24] P. Bourque, R.E. Fairley (Eds.), SWEBOK: Guide to the software engineering body of knowledge, Version 3.0, IEEE Computer Society, Los Alamitos, CA, 2014, URL http://www.swebok.org/.

[25] E.A. Lee, S.A. Seshia, Introduction To Embedded Systems, Second Edition : A Cyber-Physical Systems Approach, The MIT Press, 2016, URL https://ptolemy.berkeley.edu/books/leeseshia/.

[26] F. DeRemer, H. Kron, Programming-in-the large versus programming-in-the-small, in: Proceedings of the International Conference on Reliable Software, Association for Computing Machinery, New York, NY, USA, 1975, pp. 114–121, http://dx.doi.org/10.1145/800027.808431.

[27] C. Ebert, C.H. Duarte, Digital transformation, IEEE Softw. 35 (2018) 16–21, http://dx.doi.org/10.1109/MS.2018.2801537.

[28] S. Berghaus, A. Back, Stages in digital business transformation: Results of an empirical maturity study, in: MCIS 2016 Proceedings, AIS Electronic Library (AISeL), 2016, URL https://www.alexandria.unisg.ch/249286/.

[29] N. Bahnes, B. Kechar, H. Haffaf, Cooperation between intelligent autonomous vehicles to enhance container terminal operations, J. Innov. Digit. Ecosyst. 3 (1) (2016) 22–29, http://dx.doi.org/10.1016/j.jides.2016.05.002.

[30] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, Association for Computing Machinery, New York, NY, USA, 2014, http://dx.doi.org/10.1145/2601248.2601268.

[31] D. Horváth, R. Szabó, Driving forces and barriers of industry 4.0: Do multinational and small and medium-sized companies have equal opportunities? Technol. Forecast. Soc. Change 146 (2019) 119–132, http://dx.doi.org/10.1016/j.techfore.2019.05.021.

[32] G. Vial, Understanding digital transformation: A review and a research agenda, J. Strateg. Inf. Syst. 28 (2) (2019) 118–144, http://dx.doi.org/10.1016/j.jsis.2019.01.003, SI: Review issue.

[33] P.C. Verhoef, T. Broekhuizen, Y. Bart, A. Bhattacharya, J. Qi Dong, N. Fabian, M. Haenlein, Digital transformation: A multidisciplinary reflection and research agenda, J. Bus. Res. 122 (2021) 889–901, http://dx.doi.org/10.1016/j.jbusres.2019.09.022.

[34] K.S. Warner, M. Wäger, Building dynamic capabilities for digital transformation: An ongoing process of strategic renewal, Long Range Plan. 52 (3) (2019) 326–349, http://dx.doi.org/10.1016/j.lrp.2018.12.001.

[35] A. Singh, T. Hess, How chief digital officers promote the digital transformation of their companies, MIS Q. Exec. 16 (2017) 1–17.

[36] T. Machado, D. Fassbender, A. Taheri, D. Eriksson, H. Gupta, A. Molaei, P. Forte, P.K. Rai, R. Ghabcheloo, S. Makinen, A.J. Lilienthal, H. Andreasson, M. Geimer, Autonomous heavy-duty mobile machinery: A multidisciplinary collaborative challenge, in: 2021 IEEE International Conference on Technology and Entrepreneurship, ICTE, IEEE, 2021, pp. 1–8, http://dx.doi.org/10.1109/icte51655.2021.9584498.

[37] C. Matt, T. Hess, A. Benlian, Digital transformation strategies, Bus. Inf. Syst. Eng. 57 (5) (2015) 339–343, http://dx.doi.org/10.1007/s12599-015-0401-5.

[38] S. Liu, J.L. Gaudiot, H. Kasahara, Engineering education in the age of autonomous machines, Computer 54 (4) (2021) 66–69, http://dx.doi.org/10.1109/MC.2021.3057407.

[39] B. Siciliano, O. Khatib, Springer Handbook of Robotics, second ed., Springer Publishing Company, Incorporated, 2016.

[40] R. Siegwart, I.R. Nourbakhsh, D. Scaramuzza, Introduction To Autonomous Mobile Robots, second ed., The MIT Press, 2011.

[41] R.Y. Yates, Onboarding in software engineering (Ph.D. thesis), University of Limerick, 2014, URL http://hdl.handle.net/10344/4272.

[42] A. Ju, H. Sajnani, S. Kelly, K. Herzig, A case study of onboarding in software teams: Tasks and strategies, in: Proceedings - International Conference on Software Engineering, 2021, pp. 613–623, http://dx.doi.org/10.1109/ICSE43902.2021.00063, arXiv:2103.05055.

[43] G. Tassey, Standardization in technology-based markets, Res. Policy 29 (4) (2000) 587–602, http://dx.doi.org/10.1016/S0048-7333(99)00091-8.

[44] M. Hervás-Peralta, S. Poveda-Reyes, G.D. Molero, F.E. Santarremigia, J.-P. Pastor-Ferrando, Improving the performance of dry and maritime ports by increasing knowledge about the most relevant functionalities of the terminal operating system (TOS), Sustainability 11 (6) (2019) http://dx.doi.org/10.3390/su11061648.

[45] ISO, Organization and Digitization of Information About Buildings and Civil Engineering Works, Including Building Information Modelling (BIM) — Information Management Using Building Information Modelling, Vol. 2018, International Standards Organization, 2018, URL https://www.iso.org/standard/68078.html.

[46] S. Vivek, D. Verma, P. Krishnan, Towards solving the IoT standards gap, in: 2018 International Conference on Advances in Computing, Communications and Informatics, ICACCI, 2018, pp. 1441–1447, http://dx.doi.org/10.1109/ICACCI.2018.8554506.

[47] D. Thomas, W. Woodall, E. Fernandez, Next-generation ROS: Building on DDS [webinar].[online]., in: ROSCon Chicago 2014, Open Robotics, Mountain View, CA, 2014, http://dx.doi.org/10.36288/ROSCon2014-900183.

[48] L. Luostarinen, Novel virtual environment and real-time simulation based methods for improving life-cycle efficiency of non-road mobile machinery (Ph.D. thesis), Lappeenranta University of Technology, ISBN: 978-952-265-763-3, 2015, URL https://urn.fi/URN:xisbn:978-952-265-763-3.
[49] A. Afzal, D.S. Katz, C. Le Goues, C.S. Timperley, Simulation for robotics test automation: Developer perspectives, in: 2021 14th IEEE Conference on Software Testing, Verification and Validation, ICST, 2021, pp. 263–274, http://dx.doi.org/10.1109/ICST49551.2021.00036.
[50] H. Choi, C. Crump, C. Duriez, A. Elmquist, G. Hager, D. Han, F. Hearl, J. Hodgins, A. Jain, F. Leve, C. Li, F. Meier, D. Negrut, L. Righetti, A. Rodriguez, J. Tan, J. Trinkle, On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward, Proc. Natl. Acad. Sci. 118 (1) (2021) e1907856118, http://dx.doi.org/10.1073/pnas.1907856118.
[51] A. Afzal, Automated Testing of Robotic and Cyberphysical Systems (Ph.D. thesis), Carnegie Mellon University, 2021, http://dx.doi.org/10.1184/R1/16645639.v1, URL https://kilthub.cmu.edu/articles/thesis/Automated_Testing_of_Robotic_and_Cyberphysical_Systems/16645639/1.
[52] P. Koopman, M. Wagner, Challenges in autonomous vehicle testing and validation, SAE Int. J. Transp. Saf. 4 (1) (2016) 15–24.
[53] ISO, Road Vehicles — Functional Safety, Vol. 2018, International Standards Organization, 2018, URL https://www.iso.org/standard/68383.html.
[54] ISO, Road vehicles — Safety of the intended functionality, Vol. 2022, International Standards Organization, 2022, URL https://www.iso.org/standard/77490.html.
[55] P. Salvini, D. Paez-Granados, A. Billard, On the safety of mobile robots serving in public spaces: Identifying gaps in EN ISO 13482: 2014 and calling for a new standard, ACM Trans. Hum.-Robot Interact. (THRI) 10 (3) (2021) 1–27.
[56] ISO, Robots and Robotic Devices — Safety Requirements for Personal Care Robots, Vol. 2014, International Standards Organization, 2014, URL https://www.iso.org/standard/53820.html.
[57] ISO, Earth-Moving Machinery and Mining — Autonomous and Semi-Autonomous Machine System Safety, Vol. 2019, International Standards Organization, 2019, URL https://www.iso.org/standard/76126.html.
[58] ISO, Agricultural Machinery and Tractors — Safety of Highly Automated Agricultural Machines — Principles for Design, Vol. 2018, International Standards Organization, 2018, URL https://www.iso.org/standard/62659.html.
[59] ISO, Industrial Trucks — Safety Requirements and Verification — Part 4: Driverless Industrial Trucks and Their Systems, Vol. 2020, International Standards Organization, 2020, URL https://www.iso.org/standard/70660.html.
[60] M.L. McHugh, Interrater reliability: the kappa statistic, Biochem. Med. 22 (3) (2012) 276–282, URL https://pubmed.ncbi.nlm.nih.gov/23092060. 23092060[pmid].

Andrei Ahonen is currently a doctoral researcher at Doctoral School for Industry Innovations, Tampere University, Finland. He received his master's degree in automation engineering from Tampere University, in 2019. His research interests include software engineering for heavy duty mobile machine automation and robotics.

Aimee de Koning is currently a doctoral researcher at the Doctoral school for Industry Innovations, Tampere University, Finland. She received her master's degree in industrial engineering from Aschaffenburg university of applied sciences, in 2019. Her research interests include embedded software development, safety, and design of heavy-duty mobile machine automation.

Tyrone Machado is currently an industrial doctoral researcher at Bosch Rexroth AG, Elchingen, Germany, and is also enrolled in the Ph.D. program at Tampere University, Tampere, Finland. He received his Master of Science degree in Production Engineering from Chalmers University of Technology, Sweden in 2019. His research interests are multidisciplinary, primarily focusing on topics related to business cases, business models, and strategic management in the context of new generation automated and autonomous heavy-duty machinery.

Dr. Reza Ghabcheloo received the M.S. degree in Control Engineering from K.N Toosi Univ of Technology, Tehran, Iran in 1999, and the Ph.D. degree in Electrical Engineering from Technical University of Lisbon, Portugal in 2007 on "Coordinated Path Following of Multiple Autonomous Vehicles".

He is currently a professor with Faculty of Engineering and Natural Sciences of Tampere University. He is leading autonomous working machines research group. His research interest is at the intersections of robotics and working machines, optimization and machine learning, motion control and perception. He has over 80 journal and conference articles in these areas. His motto is practically relevant research on solid theoretical grounding. He is responsible for Robotics major in Tampere University.

Dr. Outi Sievi-Korte is currently engaged in the European Commission initiative Destination Earth, as since 2022 she has worked as the technical coordinator for the Climate DT project at CSC-IT Centre for Science. She has a background working as an Assistant professor (tenure track) in Software Engineering at Tampere University, Finland, where she was employed since 2009. She defended her dissertation in 2011, and received a Post-Doctoral Researcher grant from the Academy of Finland in 2014 for research in the field of global software development. She has published over 35 peer-reviewed papers in software engineering.