Does SoC Hardware Development Become Agile by Saying So: A Literature Review and Mapping Study

ANTTI RAUTAKOURA and TIMO HÄMÄLÄINEN, Tampere University, Finland

The success of *agile development* methods in software development has raised interest in System-on-Chip (SoC) design, which involves high architectural and development process complexity under time and project management pressure. This article discovers the current state of *agile hardware development* with the questions (1) how well literature covers the SoC development process, (2) what agile methods and practices are applied or (3) what proposals are made to increase the agility, and (4) what is the impact for the SoC community. To answer the questions, a mapping study and literature review were performed. Seven hundred thirty papers were first studied, and eventually, after a rigorous filtering process, 25 papers were thoroughly analyzed. The results show that the popular *agile SW development methods* are applied in 5 cases, ideas adapted from the *agile Hardware manifesto* in 9 cases, and 11 cases do not define the Agile HW development method. Most of the papers address shorter development time by better methodologies and tools that indirectly shape the SoC development toward agility. The focus of *agile hardware development* is mostly on the SoC artifacts and methodological improvements have not been quantified. However, the literature indicates a significant impact on many academic chip prototypes. The challenges are better understood and the interest in agile methods is clearly increasing. The methodological gaps in the prevalent situation encourage further research and more accurate reporting of the development in addition to the SoC artifacts.

CCS Concepts: • **Very large-scale integration design** • **Application specific integrated circuits** • **Economics of chip design and manufacturing** • **System on a chip**;

Additional Key Words and Phrases: System-on-Chip, methodology development, agile development, literature review, mapping study

ACM Reference format:
Antti Rautakoura and Timo Hämäläinen. 2023. Does SoC Hardware Development Become Agile by Saying So: A Literature Review and Mapping Study. *ACM Trans. Embedd. Comput. Syst.* 22, 3, Article 44 (April 2023), 27 pages.
https://doi.org/10.1145/3578554

1 INTRODUCTION

System-on-Chip (SoC) designs are no longer owned by specialized companies like processor manufacturers, but companies like Amazon, Apple, Tesla, and Google have presented their in-house developed chips. At the same time, multiple smaller companies and academia have started SoC designs that are often based on RISC-V and other open source components. SoC projects are thus greatly expanded to a much larger community and include many new developers that often

Authors' address: A. Rautakoura and T. Hämäläinen, Tampere University, Korkeakoulunkatu 7, Tampere, Pirkanmaa, Finland, 33720; emails: {antti.rautakoura, timo.hamalainen}@tuni.fi.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
© 2023 Copyright held by the owner/author(s).
1539-9087/2023/04-ART44
https://doi.org/10.1145/3578554

have more software engineering than **Application-Specific Integrated Circuit (ASIC)** design background. Therefore there is more interest and experience in agile development than before.

At the same time, SoCs have become highly varying from tiny IoT chips to very large multi-subsystem chips with the highest possible performance. The development processes should support a multitude of reusable **Intellectual Property (IP)** components, architectures, ASIC technologies, and changing versions over time. The key difference compared to a SW project is that the SoC is a physical product. It includes specific design steps that require dedicated tools and knowledge, which is not used elsewhere. The ordering of the steps is critical and causes long dependency chains. One design iteration round including only tool runtime from **Register-Transfer Level (RTL)** code to ASIC layout is measured in weeks, and the SoC project can last even three years for industry-scale devices. However, the system-level design from initial requirements to RTL code can be carried out faster than synthesis and physical design by using abstracted models. As a whole, the current situation challenges the traditional HW design approach looking for faster, more cost-efficient, and predictable SoC development, and the agile methods focus on solving such challenges.

The transition from waterfall to agile methods has been very successful in SW development. There have also been attempts to bring agile methods to **Embedded Systems (ES)** [23] as one step toward physical systems. The goal of this article is to discover how much and what kind of agile methods are used in SoC development. We present a literature review and mapping study, which is, to our best knowledge, the first of this kind.

The main contributions are (1) introduction of our SoC design process model for reference, (2) search string construction and literature scan for over 20k publications, and (3) deep analysis of 25 publications and conclusions on state-of-the-art in agile SoC hardware development.

This article is structured as follows. We first present an overview of the SoC development process in Section 2 to point out the potential space for agile development. For the clarity of the terms, we next define agile development in Section 3. The related work on similar reviews and mapping studies is given in Section 4. The applied research methodology for this article is described in Section 5. The mapping study results are presented in Section 6, and deeper analysis in the literature review takes place in Section 7. The conclusions are given in Section 9. The mapping study and the literature review data can be found in the appendix.

2 SOC DEVELOPMENT PROCESS

System-on-Chip is composed of an HW structure and SW that is executed on the programmable units. An example of a recent SoC architecture is presented in Figure 1. The SoC is composed of sub-systems, which are constructed from IP components. The design of these entities requires several design abstractions starting from high-level models and ending up with RTL descriptions in **Hardware Description Language (HDL)** for ASIC synthesis. The design hierarchy is used to manage component reuse and enable concurrent development.

SoC development is very demanding, which is why there have been attempts to model the design process. Gajski and Kuhn [19] expressed the system design as early as 1983 as a Y-chart expressing the functionality, architecture, and implementation degrees with several levels of abstraction. Keutzer et al. [28] define that the purpose of the SoC design process is to balance development costs (time and production cost) with system performance and functionality. They state that an efficient SoC design process can be achieved through orthogonalization of concerns and reusable hardware. Their methodology is known as Platform-based design.

Keating and Bricaud [26] have published a widely accepted book entitled *Reuse Methodology Manual for System-on-a-Chip Designs*. They model the SoC development process in a spiral process model and propose reusable pre-verified IP blocks to reduce the design costs. The aforementioned

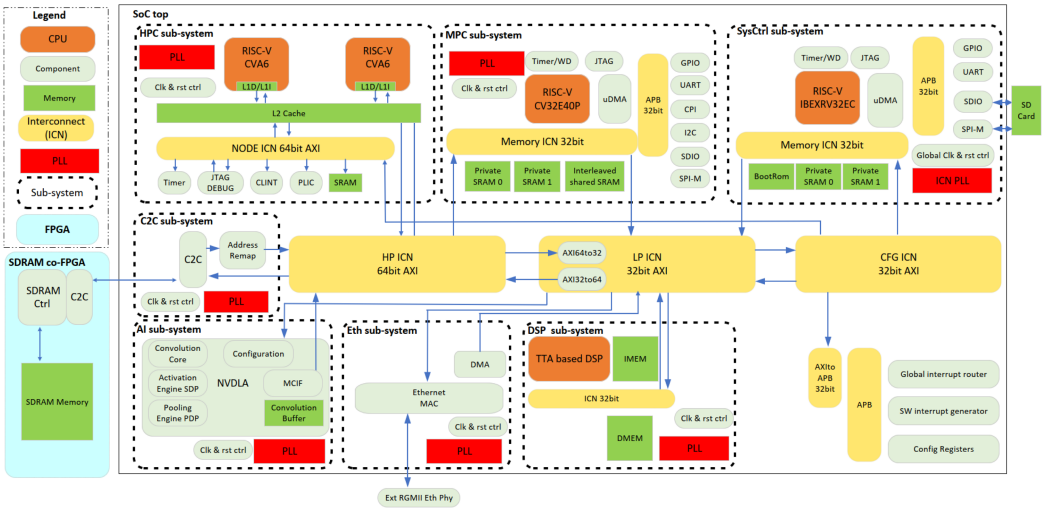

Fig. 1. An example System-on-Chip *Ballast* [37].

spiral process model was originally presented by Boehm [10] to improve the waterfall process model of the SW development. As the name indicates, the development should consist of multiple iterative rounds, each of them ending to the prototype of the SW during the development lifecycle or the SW release to the customer during the maintenance lifecycle. The spiral model has been also argued to be the process model toward agile development methods.

Although all of the presented prior works are commonly accepted for the prevalent SoC development process, we argue that none of them models the modern complex SoC design process with all of the required aspects. To complete the picture, we present a holistic SoC development process in the following. The model has been utilized in our recent SoC development project[1] with a successful tapeout of a large 15-mm^2 heterogeneous multiprocessor SoC at 22-nm ASIC technology [37].

Our SoC development process model is depicted in Figure 2. It addresses project management, lifecycles or timeline, hierarchical system composition, parallel iterative development activities, as well as dedicated development methodologies with tool flows and milestones.

We recognize seven different key *activities* involved in the SoC development. The activities are depicted in Figure 2 as horizontal arrows. They include Modeling (1), Mixed signal (combined analog and digital circuits) design (2), RTL design (3), Physical design front-end, Physical design back-end (4), Verification (5), Prototyping (6), and Hardware Dependent SW and HW/SW integration and validation (7). The activities are often expressed as **Electronic Design Automation (EDA)** tool flows, but we want to keep the principles and tools separated. However, most tools are specific to the activity, which requires dedicated developer expertise. Taken the complexity of modern SoCs leads to a project with multiple specific teams, tens to hundreds of engineering staff, and dozens of very special tools.

Milestones (M0–M9) consider the readiness of the SoC from the project start (M0) to the validated physical chip samples (M9). The milestones are used to communicate status across teams, synchronize the work between the activities, and act as quality control points. For project management, the milestones guide scheduling, resourcing, and decision making in general. The Development

[1] www.sochub.fi.

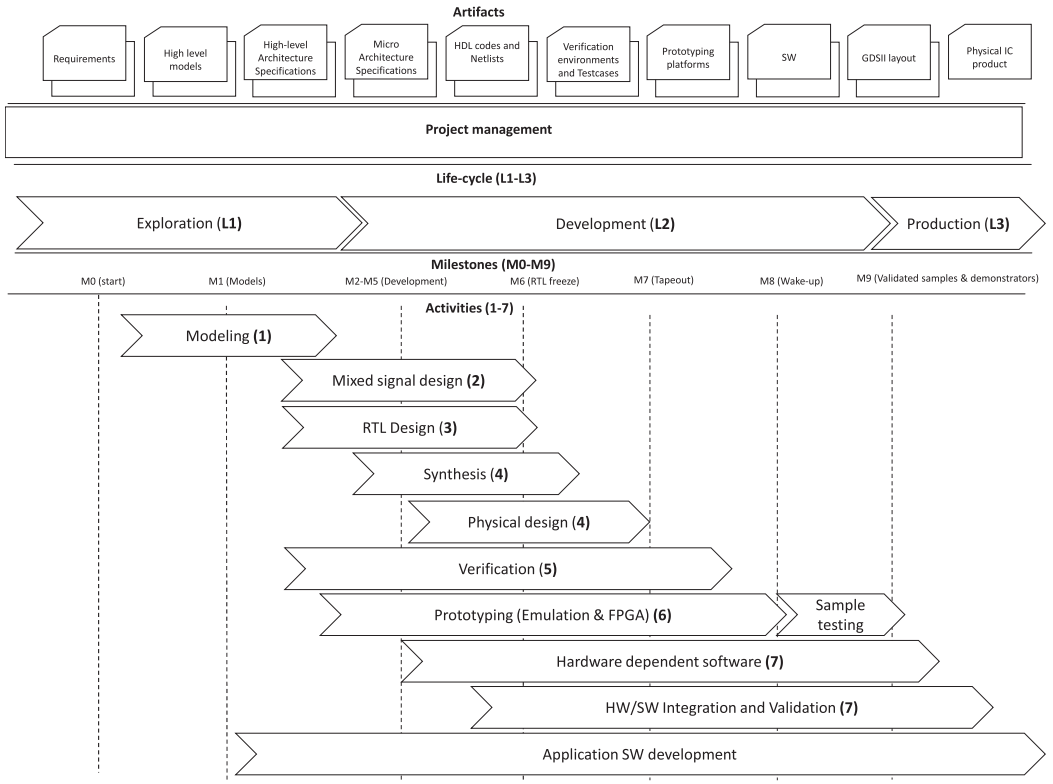

Fig. 2. SoC development process: Artifacts, lifecycle, and activities.

(L2) includes several milestones (M2–M5), which reflect the design maturity from high-level specification models to verified timing accurate chip logic and layout models. Milestones lack clear definitions in the literature, which is why we proposed one view for them in our earlier work [37].

The activities produce tangibles that we call the SoC design *Artifacts*. Only the most relevant are listed here. The artifacts from left to right in Figure 2 are Requirements, High-level Architectural Specification, High-level Models, Micro-Architectural Specifications, HDL codes, Gate netlists, Verification environments and testcases, Prototyping platforms, and SW and eventually physical layout descriptions in GDSII format for ASIC production and eventually manufactured chip itself with the test devices. The project and SoC development process is divided into Exploration (L1), Development (L2), and Production (L3) *Lifecycles* to underline the different nature of the work.

Project management is an integral part of large-scale activity. In our case, it is considered an umbrella for non-development activities, such as setting priorities, resource planning, schedule planning, team leading, quality control, and decision-making in general. The *agile development values, principles, and methods* give big emphasis on how the project management and development teams should interplay to improve productivity and predictability.

The SoC development process looks waterfall at first glance, but concurrent development tasks happen constantly among the Activities. Intermediate results of the work provide feedback for the Activities, which leads to an iterative process. The concurrency and iterations also happen between design hierarchies (System, Sub-systems, IP components), and often large sub-systems are run as separate sub-projects with their requirements, schedule, and resources. The great challenge is the dependency between the activities. That causes long feedback loops over the development

cycle. For example, the netlist as a result of synthesis activity is dependent on HDL codes qualified by verification tasks. Long feedback loops between design hierarchies can occur, for example, when system-level computation performance or power consumption is not met by underlying IP component synthesis.

Compared to SW projects the flaws are very expensive in human effort and calendar time in addition to production material preparation costs of additional chip re-spins. Re-running simulations, synthesis, or physical design can take weeks after a design change. The conclusion is that the prevalent design methodology is waterfall and improvements in the SoC design process can lead to significant economic impact. In this article, we explore the extent of agile methods for the impact.

2.1 SoC Development Methodologies

This section gives an overview of the SoC development methodologies. We will focus on those that help categorize the mapping study and literature review results. This methodology overview is written based on our research group's knowledge rooted in decades of experience in SoC design in industry collaboration and academia.

The **Model-Driven Development (MDD)** is a top-down approach. The design starts from abstract models and the flow continues through multiple intermediate modeling layers toward the target implementation. Each of the modeling layers adds more implementation-specific details to the model. MDD attempts well-defined model semantics, which enables correct-by-construct practices and formalism for code generation. The **Unified Modeling Language (UML)** is well known to model SW architectures before implementation. UML has also been extended to HW modeling with specific domain-specific metamodels and profiles. The IP-XACT standard is a modeling language for SoC development, and there are IP-XACT development tools such as Kactus2 [24]. In our previous work, we have demonstrated a multi-layer MDD framework for SoC HW development [38].

The methodologies, or practices, related to MDD-based SoC development are *abstraction*, *model transformations*, *code generation*, and *synthesis*. The abstraction is present in the *modeling* activity (1) in Figure 2 producing High-level Models as an artifact. *Model transformation* typically means automatic transformations between models. SW compilation with a compiler is a well-known example of model transformation. *Code generation* can be thought of as the last level transformation to the target artifact such as the Verilog RTL model. *The synthesis* is also one form of model transformation. The key difference is that the synthesis tool produces output by solving the given constraints. For example, the synthesis from Verilog to netlist based on given target frequency constraint. **High-level synthesis (HLS)** from C++ to Verilog is an example of source code transformation.

The development platform, or development framework, is a collection of tools, repositories, and communication and documentation facilities for carrying out design tasks in a team. We define the development platform so that it must include multiple design activities and not focus only on a single methodology.

The HW-SW co-design methodology focuses on efficient design exploration between two implementation alternatives. The work with HW-SW co-design flows start with an abstract model of the application, after which the goal is to find an optimal HW-SW division under given constraints. Executable system models written in programming languages are typically used, because the exploration needs measurable results. The HW-SW co-design can be regarded as MDD practice, but the latter focuses mainly to design exploration.

The open source hardware is not a design methodology in the traditional meaning, but we have included it due to the significant boom after releasing of the open RISC-V **instruction set architecture (ISA)** and its open source core implementations. Open source HW can mean open

license-free standards, programming languages and libraries, design tools and tool flows, HW component libraries, or even open ASIC technology libraries. Open source promises transparency, easier collaboration, and reduced costs, but from the development methodology point of view, the key question is the ease of reuse.

The term *platform architecture* related practices were shortly described by Keutzer et al. [28] and mentioned here for later reference. As we notice, all these methods have similarities and are targeting to increase the productivity of the SoC development. A common nominator is that they are focusing on the exploration lifecycle of the development. For this reason, the methodologies are often combined with traditional synthesis and physical design tools and methodologies. The presented methodology definitions are used later in this mapping study and literature review to categorise the results.

3 AGILE DEVELOPMENT

The *Agile Software Development* defines four well-defined values and 12 principles in the famous *Manifesto for Agile Development* [9]. That was a response and a criticism toward the traditional software development. The defined values are (1) *individuals and interactions over processes and tools*, (2) *working software over comprehensive documentation*, (3) *customer collaboration over contract negotiation*, and (4) *responding to change over following a plan*. The given principles are the second layer of the manifesto, but still those remain very abstract and do not provide practical guidance to implement them. Unfortunately the values and principles may not success defining agility and have led to various interpretations [30]. Thus, Laanti et al. [30] propose to focus on a different agile practices and benefits of them instead of speaking generally about the agile development.

Agile practices can be seen as a third layer of abstraction. They give more detailed guidance about how different values and principles could be achieved in practice. In the literature, the practices are also referred to as methodologies, but we rather use the term practice in this context to avoid confusion with the SoC development methodologies.

Scrum and **Extreme Programming (XP)** are the most well-known set of practices for running agile SW development projects. Scrum focuses to project management practices, and XP focuses mostly on the practical guidelines for software development such as **Test-Driven Development (TDD)** and **Continuous Integration (CI)** [2]. There are also other documented practices such as Crystal, Adaptive System Development, and Dynamic Systems Development Method, but the popularity of the deployment or related research for those seems not to be on the same level as it is for SCRUM and XP [17].

The early work of *Agile Software Development* focused on small-scale software development and the practices were addressing team-level work. As a consequence, large-scale agile has also been addressed widely [15]. Scaled agile includes framework proposals such as Large-scale SCRUM, Scaled Agile Framework, and SCRUM-of-SCRUMS and practices such as Agile Release Train [34]. Definitions of the project scale for large-scale projects measured with the amount of organization involved or engineering headcount varies, but in contrast non-large-scale agile practices often refer to engineering headcount from 8 to 15 who are responsible for the complete software project.

Tailoring of agile development practices is common for different domains and types of projects. Kaisti et al. [23] notice the importance of process tailoring for embedded system development. There has been also an effort to establish principles and practices for agile SoC development by UC Berkeley [31] (2016). Their proposal includes the following four principles: (1) *incomplete, fabricable prototypes over fully featured models*; (2) *collaborative, flexible teams over rigid silos*; (3) *improvement of tools and generators over the improvement of the instance*; and (4) *response to change over following a plan (Iterative design)*. In this article, we call their proposal an *agile HW*

manifesto. We regard it as the best-known formalized agile method in SoC development due to its well-documented definition. When we compare these principles to the values of the *Manifesto for Agile Development* we notice similarities but also an important difference. They contradict clearly in the importance of the tools. Schrof et al. [40] also find out that when deploying agile approaches to mechanical HW development, there are needed specific methodologies. They call them *technology enablements* that are needed to overcome constraints and the nature of the development of the physical product.

Agile practices on SW development have become possible due to widely available tools. For example, TDD benefits from a vast amount of free Unit Test Frameworks, such as the JUnit for Java or Gitlab CI, to execute tests constantly in an automatic manner. Technologies are also seen as enablers to apply agile methods in automotive [40] where the physical end product complicates the use of existing methods as such. Due to these observations and the young age of agile HW development, we look for applied tools and technologies in this work.

Despite the popularity of agile SW development, there are notable shortcomings in defining agility. In addition, there is a lack of empirical studies to measure the effectiveness of the agile methods [1]. As we are facing the same issues in agile HW development even on a larger magnitude, it is too early to provide quantified measures as the outcome of this study.

In summary, despite the many descriptions, definitions, and agile SW development articles there is no single reference that would explain it thoroughly and cover the full development view. However, the common themes can be recognized such as the importance of the people, the presence of the customer, short development iterations, fixed cadence of the releases (schedule is prioritized over content), and constant improvements of the working practices.

4 RELATED WORK

As far as we know there are no prior mapping studies or literature reviews focusing on agile SoC HW development. Work done by Kaisti et al. [23] focuses on agile methods for embedded systems and embedded SW development. The primary result is that the embedded systems domain includes problems that need to be solved before agile methods can be successfully applied. Reported challenges were a lack of tools, real-time constraints of the embedded systems, a need for HW-SW co-design, and a need for documentation to distribute information across different design disciplines. However, they also find common themes in successful practices that include test-driven development, continuous integration, dual targeting (simulation before actual hardware), iterative development, and customer collaboration.

The literature on embedded systems includes proposals on how to increase the agility of the embedded SW and the system development with methodologies such as HW-SW co-design [41] and platform-based design [12]. The proposed methodologies are relevant also for SoC development. Kaisti et al. also discuss embedded HW and ASICs, and they note that very little is known about agile HW development as peer-reviewed academic work and there is a need for more rigorous research.

Demissie et al. [14] focuses on safety-critical embedded systems. The work shared eight references with Reference [23] and included 14 papers published after it. The results show challenges such as HW development, team-based communication, and regulation process. Studied works applied selected agile practices mainly from SCRUM and XP. Some of these included studies addressed the need for tools support to effectively implement agile practices in the context of embedded systems.

The review by Ahmad [3] covers agile development of **Cyber-Physical Systems (CPS)**. This work focuses on the presence of agile development in the CPS domain. The result is that related literature is limited, but the trend indicates an increasing interest.

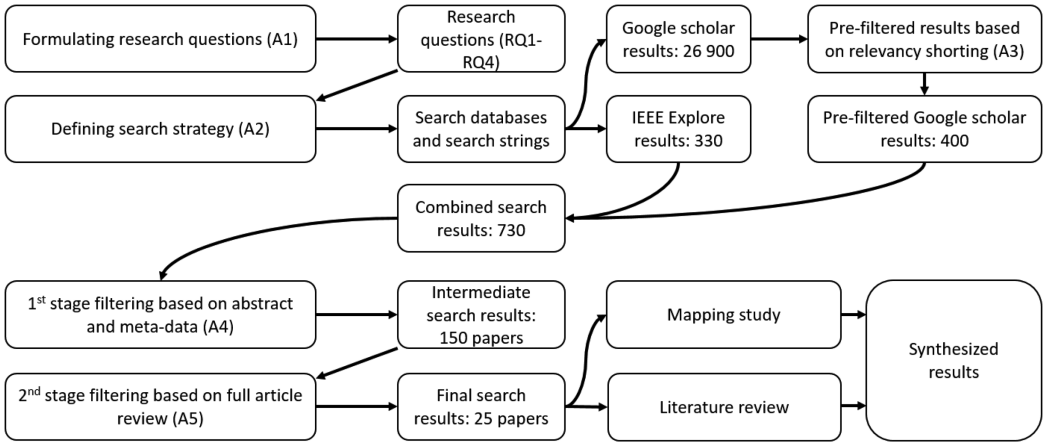

Fig. 3. The mapping study and literature review process overview.

To conclude the related work, SoC HW development has not been directly addressed in the prior literature reviews. Earlier works, however, discuss challenges and solutions relevant to the SoC HW development.

5 RESEARCH METHOD

The research methodologies used in this study are systematic mapping study and systematic literature review defined by Keele et al. [27]. The purpose of the mapping study is to show selected attributes among publications to provide an overview. For more detailed insight, the publications are analyzed thoroughly. The systematic process is described in Figure 3.

The study starts with the activity of formulating research questions (A1) in Figure 3, and as an outcome we came up with four different research questions (RQ1–RQ4) as follows:

- RQ1: *What is the SoC HW development process coverage in the agile SoC HW development literature?*
- RQ2: *What agile methods or practices are currently applied to SoC HW development?*
- RQ3: *What methods and practices are proposed to increase the agility of the SoC development?*
- RQ4: *What is the impact of the work for the SoC HW development?*

RQ1 considers how much the literature covers the agile methods from the whole SoC HW design process. The key difference between RQ2 and RQ3 is that the former seeks the usage of known agile methods while RQ3 addresses the SoC development method proposals to increase agility. We noticed that it is important to make this separation due to the early age of the agile HW development and not yet established terminology. The search strategy (A2) and filtering steps (A3–A5) are described in the following sections.

5.1 Search Strategy

The purpose of the search strategy is to increase repeatability and quality of the work. The main task is to select search engines and databases and define the exact search string. The literature review was performed by using two search engines, Google Scholar and IEEE Explore. Google Scholar was used, because it performs the search from multiple databases. IEEE Explore was used to make sure the other search engine results and get focused results.

Based on trial searches, we end up using the search string: *agile AND ("integrated circuits" OR ASIC OR FPGA OR SoC OR "system on chip" OR "system-on-chip")*. The term "agile" was selected instead of known agile methods in SW engineering, e.g., SCRUM. We noticed that including only the known agile methods led to exclusion of many HW-oriented papers. This is explained in more detail in the mapping study results.

The search is limited to the full years 2000–2021. The searches were performed in the browser's private mode to avoid history and other metadata affecting the results. The accuracy of our approach was verified by manually checking that a set of known publications were found in the search.

Google Scholar yielded 26,900 search hits, and IEEE explore gave 330 hits. The remarkable difference is explained by the difference in the search algorithms. Google Scholar seems to be a heuristic. In practice, it means that after a certain point the results do not anymore represent the search string accurately enough. However, 26,900 is not a feasible amount to be explored even at the title level. Due to that an additional pre-filtering step (A3 in Figure 3) was performed for the Google Scholar results. The results were checked in the relevancy order shorted by Google Scholar, just by reviewing the search results (title and context of the string match). The first 400 search results were checked and included to further review. After that point, the relevancy shorting did not produce any more valuable publications for our purpose.

At this point, the combined results from both databases included approximately 730 publications.

5.2 Filtering

The filtering of the search results was performed in two stages (A4 and A5 in Figure 3). The first stage filtering was based on title, abstract, and publication metadata such as conference or journal name, while the second stage filtering was based on the full article review. Both filtering stages used the process of inclusion and exclusion [27].

The paper was included when all of the following criteria were met:

(1) The paper addressed agile development of the SoC HW.
(2) The paper was published in a peer-reviewed academic forum.

The exclusion was done if any of the following criteria were met:

(1) SoC development was not addressed.
(2) The agile development was not addressed.
(3) The agile SoC development proposal was too specific to a certain type of designs or tool flows.
(4) Non-peer reviewed or non-academic publication.
(5) Duplicate of the same work.

The inclusion and exclusion set a clear scope for this work, increased robustness, and enabled us to address the research question better. However, drawing the line on a decision for including publication to further analysis is not trivial. Examples of such borderline exclusions were deep learning accelerator design flow specific for one type of neural network, and FPGA-based cloud computing accelerator specific for a dedicated big-data tool flow. Embedded systems and embedded software-focused papers were the most common reason for exclusion. SoC HW was often mentioned in these papers, but the focus was not on the SoC HW development itself.

As discussed, the term "agile" is a challenging search string. It was often referred to as a broad term, and we need to interpret the meaning. Typical exclusions were agile as a buzzword with a

Table 1. Mapping Study Attributes and Abbreviations

Attribute	Description, abbreviations
A1: Index	The index (P1–P25) distinguishes literature review publications from other references.
A2: Year	The year of publication.
A3: Publication engineering field	The primary field of the publication forum: System (**Sys**), Hardware (**HW**), Software (**SW**)
A4: Target platform	Target implementation platform of HW. Application-Specific Integrated Circuit (**ASIC**), Field-Programmable Gate Array (**FPGA**), integrated circuit (**IC**), simulation (**SIM**), embedded system (**ES**).
A5: SoC project management and lifecycle level coverage	Which parts of the SoC process the publication address. Project management (**PM**), Exploration (**L1**), Development (**L2**) See Figure 2
A6: SoC development activity coverage	Modeling (**1**), mixed signal design (**2**), RTL design (**3**) Synthesis and physical design (**4**), Verification (**5**), Prototyping (**6**), Hardware dependent SW (**7**). See Figure 2
A7: Applied agile methodology or practices	Recognized agile methods and practices. Also proposals of the complement agile methods for SoC development are listed here.
A8: Proposed methodology or practices to increase agility	The attribute is synthesized from publications. Because terminology varies the naming has been harmonized. The key practices include: Code generation, Model transformations, Synthesis, Rising abstraction, Development platform, Platform architecture, HW-SW co-design, Open source or open access.
A9: Scale	Small (**S**), Large (**L**). Determined by the size of the development team.
A10: Evaluation method, institution and prototype implementation technology	Proposal (Prop), i.e., not prototyped as a physical chip, Case study in: Academic (**CSA**), Industry (**CSI**), or Mixed (**CSM**) setting). Implemented prototype **FPGA** or **ASIC**.

Attribute labels A1–A10 matches to columns on Tables 3 and 4.

null meaning or an agile product feature instead of agility of the development process. For example, "a frequency-agile radio SoC" without any agility is mentioned in the design flow.

As an outcome, we had 150 papers after the pre-filtering and the first stage of filtering. Eventually, 25 papers were included after a full article review (A5). The selected papers are listed in Tables 3 and 4.

5.3 Mapping Study and the Literature Review Attributes

The research questions are addressed by the mapping study and the literature review. The data were organized by attributes defined in this section. The mapping study gives an overview of the results while the literature review addresses research questions in more detail. The same publications and research questions are used for both methods.

The publications and the mapping study data are collected in Tables 3 and 4. The explanation of the attributes A1–A10 and abbreviations applied in the mapping study are given in Table 1. In all columns, "NA" stands for "not addressed." The publications are indexed by attribute A1 in alphabetical order according to the first author.

The year of publication (A2) and publication forum gives insight into the relevancy of the topic. For the mapping study, we recorded the engineering field (A3) of the forum to see in which fields the agile HW development gets published.

The target platform (A4) lists all the implementation technologies the publication is addressing. One publication can cover multiple technologies. The SoC HW can be implemented with FPGA and ASIC technologies, and the SoC can also be addressed as part of an ES. **Integrated circuit**

(IC) is a typical term for mixed-signal design. As presented in Figure 2, the executable models are relevant and can be synthesized to FPGA and ASIC technologies. Thus **simulation (Sim)** models on the SoC domain can be seen as a relevant intermediate target platform. The ES and IC attributes were included, because those are present in a few cases when the topic crosses our focus of agile HW development.

The SoC development process coverage of the publications is captured with attributes A5 (Management and lifecycle coverage) and A6 (Development activity coverage). These attributes address the research question RQ1. Due to the large scope of SoC development, it would be unlikely that a publication would address all aspects. Our interest is in observing where the current research is focused and on trying to find explanations for it.

The applied known agile methods and practices are recorded under the attribute A7. Known methods and practices originate from SW engineering, but we decided to make one exception here. The agile HW manifesto [31] is treated as a known methodology, because it has been well recognized. This attribute addresses the research question RQ2.

Attribute A8 captures the methodology or practices to increase the agility of the SoC development. The purpose is to see what literature proposes to improve the situation. This attribute address the research question RQ3.

The impact of the work is evaluated by attributes (A9) and (A10). The scale of the project (A9) measures the development organization size. We defined the scale so that a small-scale project includes a single team of up to 10 members. Large-scale projects consist of multiple teams or large teams beyond 10 members. The attribute A10 records the evaluation method, the institution type where the work has been performed, and the prototype implementation technology.

The reasons for selecting attributes A9 and A10 to judge the impact are as follows: (1) The large project setup resembles a better SoC project setting with a complex design with multiple development activities and (2) fabrication of physical chips guarantees that the design has gone through the complete SoC development process. Contributions with the hypothesis about the agile HW development without a use case are listed as proposals. Case studies performed by different types of institutions consider the development project with measured results. The research setup is captured from the reported affiliations. The key interest in the use case type is to see the industry interest or if the methods are applied in the industry. The purpose is not to grade the impact between academic and industry work. Instead, the fabricated prototype gives a clear measure of the impact. The fabricated chip is regarded as a high-impact work.

6 MAPPING STUDY

The mapping study data are illustrated in Figures 4–10 where the attributes are on the X axis and the frequency of the attribute is on the Y axis. The mapping study results are presented by data recorded in Tables 3 and 4.

6.1 Overview

The timeline of the publications is presented in Figure 4. The peak for 2020 is explained mainly by a single special issue on agile hardware [21].

The publication forums and engineering fields are listed in Table 2. We can notice highly ranked journals and conferences focusing on SoC designs being well presented. A clear majority of the publications have been published in HW forums.

It can be concluded that the interest is increasing in agile methods on SoC HW.

The target implementation HW platform is presented in Figure 5. A single publication can cover multiple technologies. ASIC, FPGA, or HW simulation were the primary target platforms. ES and

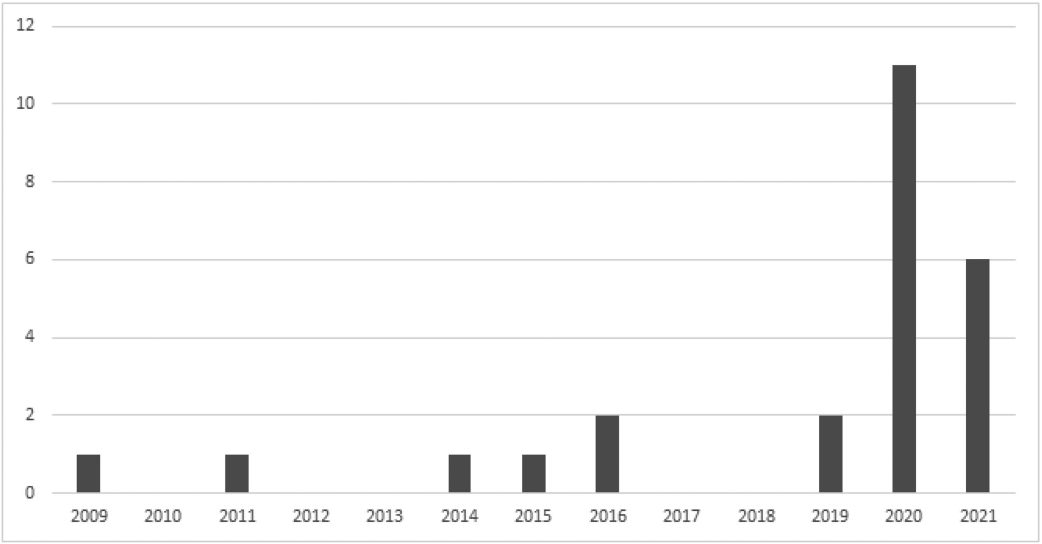

Fig. 4. The timeline and the amount of publications (A2).

Table 2. Publication Forums and Engineering Discipline (A3)

Forum	Field	Amount
IEEE Micro	HW	8
IEEE/ACM International Conference On Computer Aided Design (ICCAD)	HW	3
IEEE Journal of Solid-State Circuits	HW	2
Design Automation Conference (DAC)	HW	1
IEEE International Conference on Electro/Information Technology	HW	1
IEEE Design & Test	HW	1
IEEE Nordic Circuits and Systems Conference (NORCAS)	HW	1
IEEE/ACM International Symposium on Microarchitecture	HW	1
IEEE International Conference on Microelectronics (MIEL)	HW	1
IEEE Computer	HW	1
Conference on Programmable Logic (SPL)	HW	1
Workshop on Domain Specific System Architecture (DOSSA-3)	System	1
Journal of Systems and Software	System	1
IEEE Software Testing, Verification and Validation Workshops (ICSTW)	SW	1
International Workshop on Rapid Continuous Software Engineering	SW	1

IC domains were also present. These are relevant for SoC development and improve the reliability of the following analysis.

6.2 RQ1: What Is the SoC HW Development Process Coverage in the Agile SoC HW Development Literature?

The SoC process coverage data is presented in Figures 6 and 7. A single publication can cover multiple process elements. From Figure 6, we see that Exploration (L1) and Development (L2) lifecycles dominate the results, and **project management (PM)** has low coverage. This is surprising,

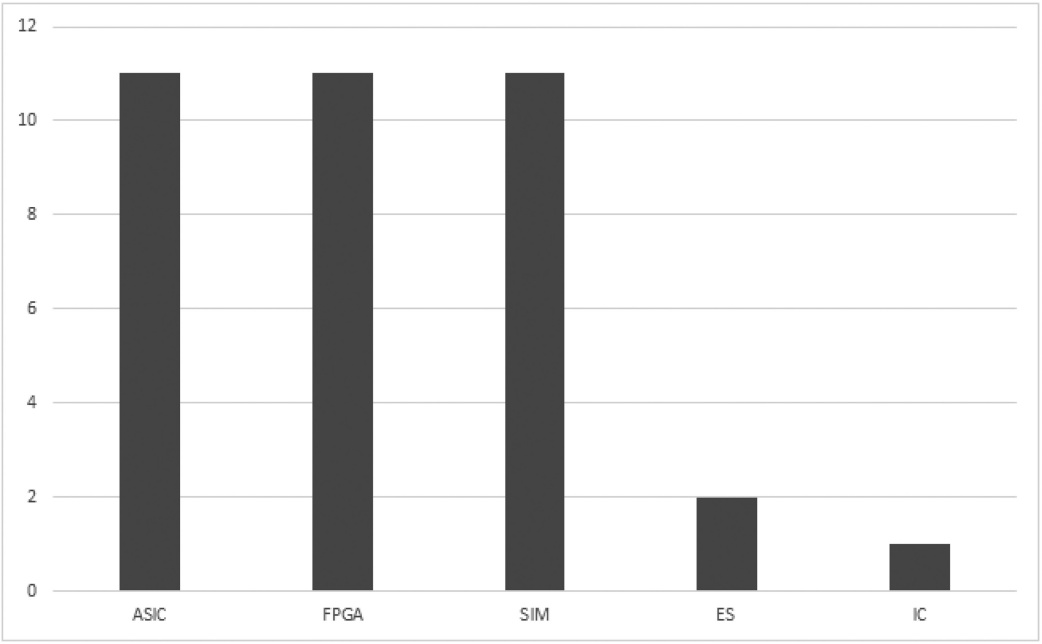

Fig. 5. Target implementation platform of the hardware (A4).

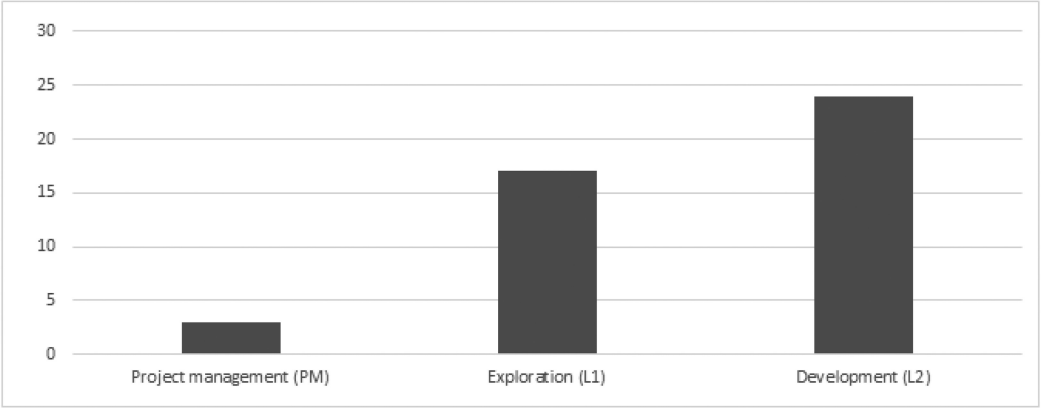

Fig. 6. SoC process coverage (A5): Project management and lifecycles L1 and L2.

because the most common agile methods (Scrum and XP) in SW engineering are project management oriented.

When we take a closer look at different SoC development activities in Figure 7, we see that all activities have been covered at least to some extent. The modeling and the RTL design activities get most of the coverage. This might indicate that rising the design abstraction from traditional RTL models to high-level models is seen as attractive and feasible. The synthesis, physical design, and prototyping activities have moderate coverage, although these are very essential for SoC development in general. The results in verification are a bit surprising. Verification is commonly using programming languages or patterns such as C or the object-oriented flavor of the SystemVerilog

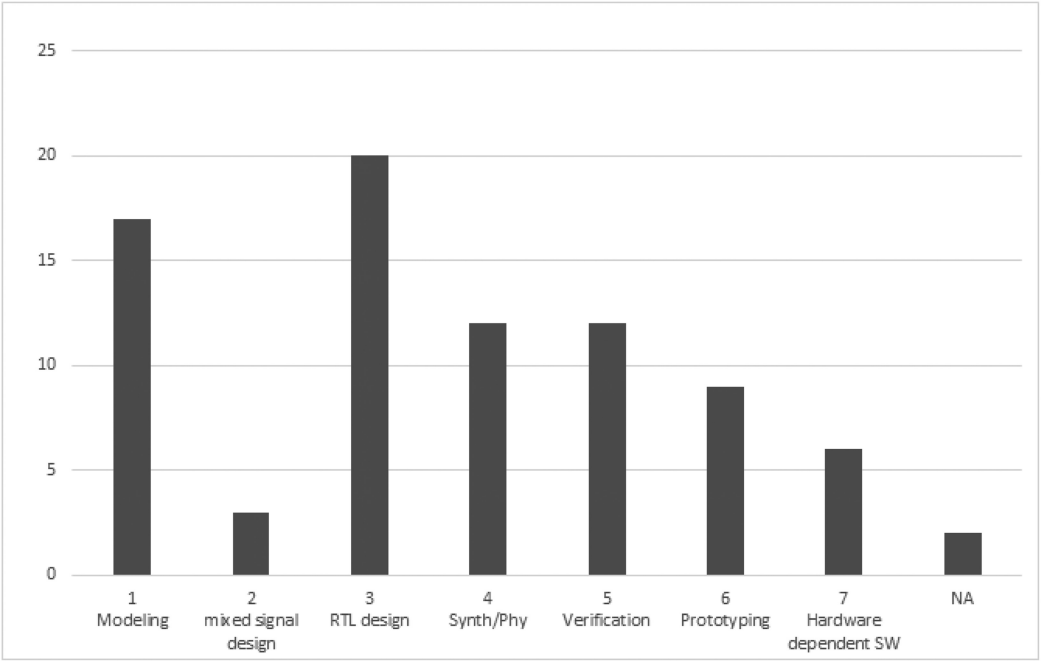

Fig. 7. SoC development activity coverage (A6).

HDL. Thus, we would expect much more agile practices. The explanation can be the long feedback loops: RTL simulation is much slower compared to SW execution.

6.3 RQ2: What Agile Methods or Practices Are Currently Applied to SoC HW Development?

Figure 8 depicts applied agile methods and practices. The results show clearly that the traditional SW-rooted agile development methods and practices have not been applied to SoC HW development to a large extent. The reason for that is unknown, and more research is needed to study this observation. The agile HW manifesto [31] seems to be currently the most famous but has not been published outside UC Berkeley. We also noticed that agile HW development has not been defined rigorously in most of the literature. Together, the results indicate that agile HW development is still in a very early phase. They did not either report problems or unsuitability of the methodology.

6.4 RQ3: What Methods and Practices Are Proposed to Increase Agility of the SoC Development?

Figure 9 lists various development practices and methods that have been proposed as a way to increase the agility of the development. The terminology and definitions vary between contributions, but conclusions are still possible from the collected data.

The most popular methodology or practice was related to MDD. Abstraction, code generation, model transformations, and synthesis were proposed as a way to increase the efficiency of SoC development. Rising the abstraction is essential to implement efficient code generation, model transformations, or synthesis. In a few cases, the increased abstraction was applied without automatic model refinements. It was seen as beneficial alone, because such models give early feedback at the

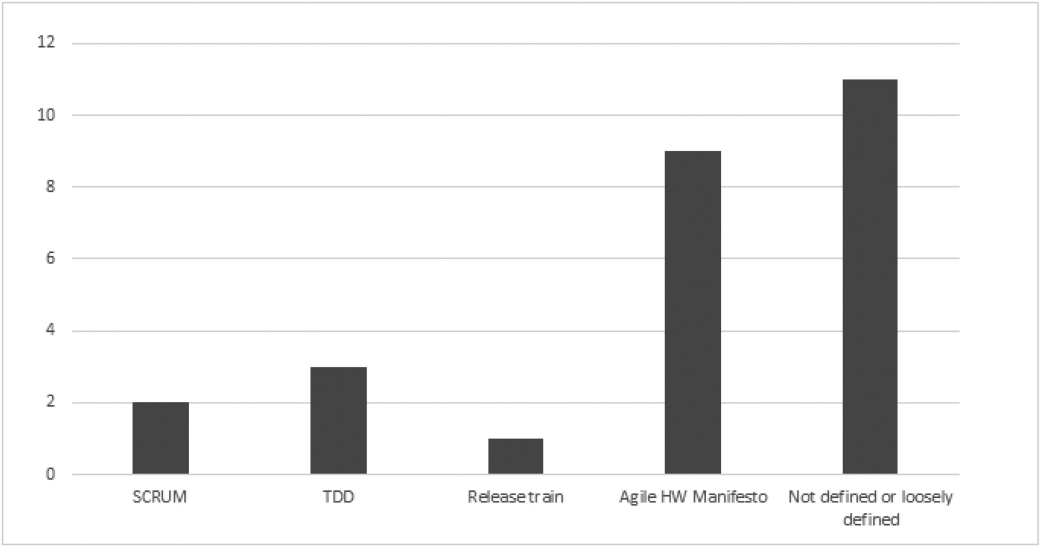

Fig. 8. Applied agile development methodologies and practices (A7).

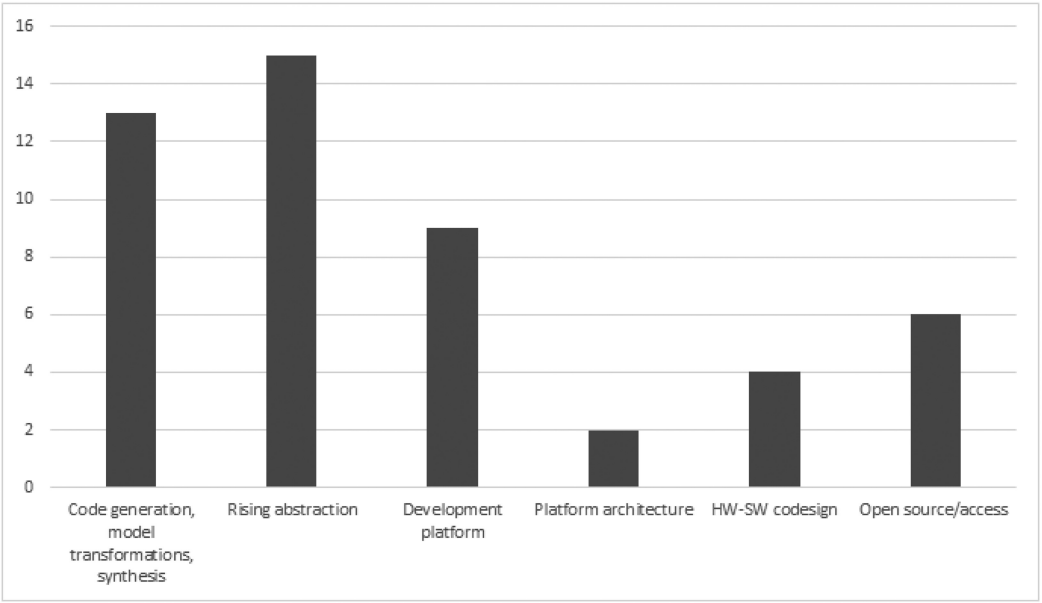

Fig. 9. Proposed methodology or practices to increase agility (A8).

beginning of the development process on the known cost of accuracy. Other proposed practices included development platform, platform architecture, HW-SW co-design, and open source components. The proposals are analyzed in more detail as part of the literature review in Section 7.2.

The important result here is that none of the proposals are novel in the SoC HW domain and, none of them are agile practices as such. The results indicate that the proposed methodologies and practices together with the agile mindset are seen as an enabler for agile HW development.

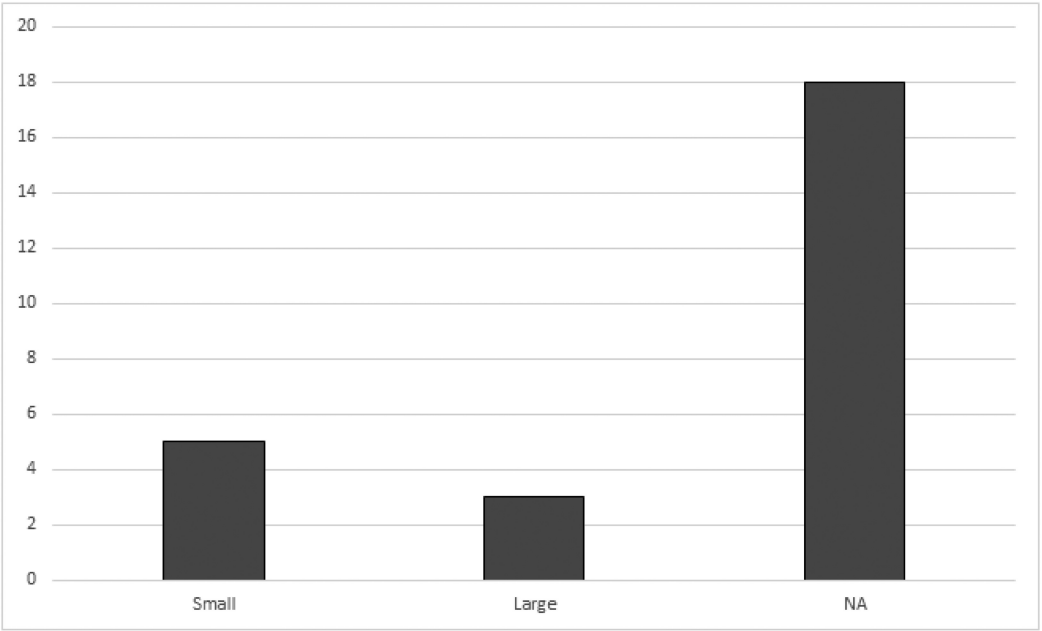

Fig. 10. Scale of the case study (A9).

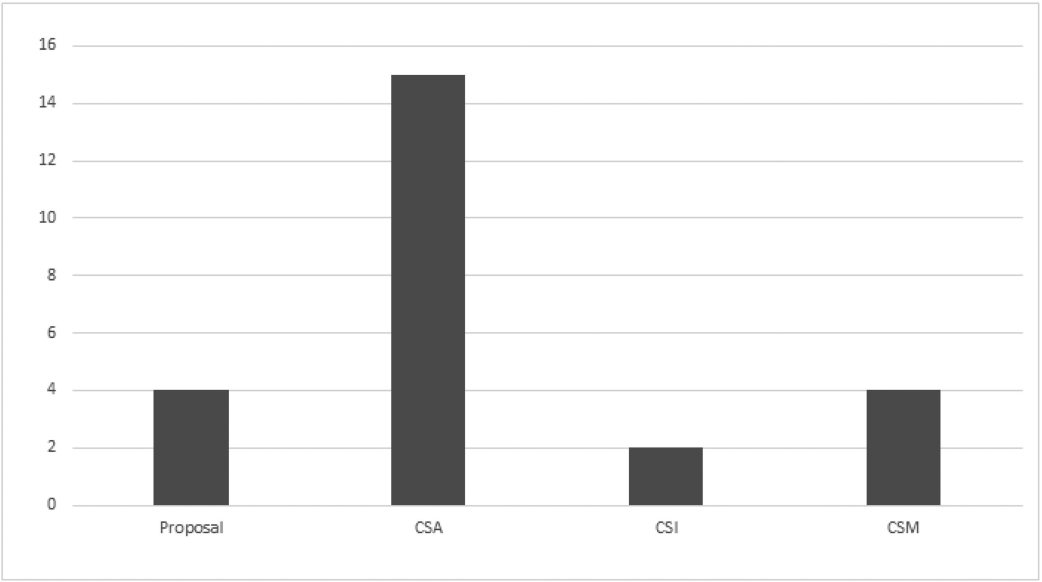

Fig. 11. The evaluation method (A10).

6.5 RQ4: What Is the Impact of the Work for the SoC HW Development?

The impact of the work was evaluated on three metrics: the scale of the work (Figure 10), evaluation method (Figure 11), and prototype technology (Figure 12).

The scale of the work was not typically mentioned, and the team was a small development team or a research group. Three were regarded as large-scale work. Only one reports the effort as 14,000

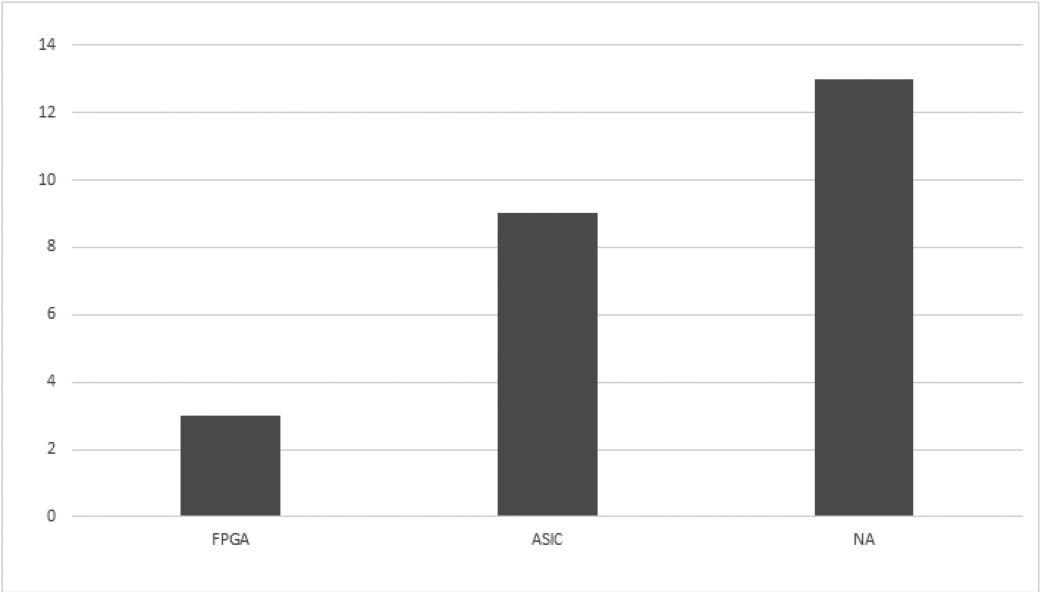

Fig. 12. Prototype implementation technology (A10).

engineering hours [7], which is close to 7 engineering years. We counted that as large-scale work, although the team size was not reported.

Case studies were the most common research setup. The industry is contributing to research mainly through collaboration with academic partners.

ASIC prototypes were reported in nine papers. One reports 11 different chip prototypes [31], but our study regards it as one, because the details of these prototypes are not shown.

In summary, the literature focuses on small-scale work in the academic setting, and the research favors practical work with prototypes over hypothetical proposals to follow the agile practices "working prototype over fully featured models." It is very positive to see so many ASIC prototypes that include the full design flow. It is well understood that those give the best impact even though ASIC prototypes are costly and labor intensive.

6.6 Summary of Results

The mapping study gives an overview of the situation and address the defined research questions at high level. The number of publications is gradually increasing. Contributions are mainly reported on HW research forums, and highly ranked journals and conferences were present in a significant extent.

The SoC design process was covered with healthy distributions and exploration and development lifecycles gets most of the coverage. The lack of project management perspective is a clear gap. We also did an observation that the more activities focus on accurate models beyond RTL the less we get coverage in the literature. We believe that this is due to long runtimes and feedback loops in the tool flows that disfavor agile practices on quick iterations.

The agile HW manifesto, SCRUM, and TDD were the most commonly known agile methodologies. However, in most of the cases the agile methods or practices is not defined, and popular methods on SW engineering such as SCRUM and XP were not present on significant extend. This can indicate that either HW developers are not familiar with the existing project management methods or those are not seen feasible.

There are many proposed methods and practices to increase agility, but none of them is completely novel or agile as such. Most of the works focuses on increasing the development efficiency with MDD practices (abstraction, code generation, model transformations, and synthesis). This is an important finding and may enable connecting SoC HW development and agile development.

The impact of the contributions is promising. ASIC prototypes are common and indicate that research is looking for practical solutions. Project organisation, team sizes, or other measures about the complexity were not typically presented, and most of the contributions were done in academic setting.

The research about agile HW development is clearly at early phase. Agile HW development was not defined clearly enough in most of the publications. However, the results show challenges in the traditional development methods and waterfall-driven processes. To improve the situation, rigorous research is needed to study the identified gaps.

7 THE LITERATURE REVIEW

This section includes detailed analysis of the publications in the form of literature review. We will focus especially on the research questions RQ2 and RQ3, because RQ1 and R2 were well covered by the mapping study. Because none of the selected publications provide quantified measurements of the design flow productivity, our contribution is to provide insights to the current state on detailed level.

7.1 RQ2: What Agile Methods or Practices Are Currently Applied to SoC HW Development?

The SCRUM was the only applied methodology [29, 44]. Reference [44] presents a tailored SCRUM proposal for Cyber-Physical Systems and covers SoC hardware development by proposing (but not implementing) simulation models of the HW for fast prototyping and HW-SW co-design. The key concepts discussed are separated HW and SW sprints and integration of them with Agile Release Trains. As a result, the paper highlights domain specific challenges such as HW-SW co-design, architecture, model-based development and verification, which could be addressed with the proposed agile methods.

Könnölä et al. [29] apply SCRUM to embedded system and radio frequency–integrated circuit projects in the industry. Reported tailoring needs highlight challenges on the slow nature of the HW development and the different knowledge between developers in their disciplines. Different knowledge base makes the circulation of the tasks difficult, but teamwide planning sessions helped in understanding different areas. They did not increase productivity in the short term, but the increased team and system-level understanding could help productivity in the longer run.

Due to the nature of SCRUM, References [29, 44] naturally focus mainly on project management and organizational topics. Highlights of the domain challenges are useful for applying agility to the SoC HW process.

The most common agile practice was TDD [13, 18, 20]. To enable TDD for device driver development, Reference [18] showcases the usage of an abstract model of the HW by extending QEMU- and KVM-based HW virtualization software. Memory-mapped Input-output, DMA, and interrupts features were developed to increase the suitability of the development platform for TDD usage. The framework also works as an early HW model for rapid prototyping.

Jiang et al. [20] introduces PyMTL3, an open source python framework for hardware modeling, generation, simulation, and verification. The PyMTL3 claims to enable the TDD by adopting the open source pytest framework designed for unit testing of the python SW. The PyRTL [13] implements TDD by providing a toolkit where design and simulation (verification) happens in the same

environment to avoid different tooling between these activities. PyMTL3 and PyRTL frameworks are discussed in more detail in Section 7.2.

TDD is approached in a similar way in References [13, 18, 20]. The abstraction of the clock cycle–accurate RTL model to a more inaccurate high-level model, or the level of the modeling language, e.g., Verilog to python was used to enable TDD. Abstracted high-level models are faster to simulate, and led to faster iterations, and the use of programming languages for HW modeling enables the use of popular SW development TDD tools for SoC development.

Lee et al. [31] promote free and open source RISC-V, Chisel Hardware Construction Language, and hierarchical and automated physical design flow as practices to support the agile HW manifesto. An alternative implementation of the manifesto is in Reference [20] in the form of python-based modeling, generation, simulation, and verification.

The agile HW manifesto has gained much attraction despite its young age. Most importantly, multiple chip projects have been carried out. Lee et al. reports 11 chip projects, Reference [7] reports large-scale (14,000 engineering hour project) usage, Reference [25] reports a case study in a joint academic-industry setting, and [39] reports eight-core multiprocessor system with modern 16-nm FinFET technology. Reference [12] reports usage in industrial organizations and also outlines additional guidelines borrowed from the agile SW development community: (1) complexity is your enemy, (2) do not fear refactoring, and (3) do not over-engineer.

Lee et al. [31] can be regarded as the key reference due to its proven impact and strong agile mindset with defined principles. However, our earlier research [37] indicates that similar efficiency and project execution time can be achieved by constructing agility to the top of the more conventional RTL to netlist methodology instead of novel modeling languages.

Petrisko et al. [35] provides alternative principles for the HW community (be tiny, be modular, and be friendly (approachable)) based on their experience on how to increase agility. They also define four metrics to guide toward principal targets: quality, virality, efficiency, and functionality. *Software engineering of the hardware* is also mentioned as the mindset to build agile HW development. Although these principles are defined, they are rather abstract compared to the agile HW manifesto.

In a summary, the usage of known agile methods and practices in the HW domain is rare. The literature highlights challenges in adopting methodologies such as SCRUM, but TTD as a development-oriented practice can be seen as more feasible and beneficial. Early work of forming a definition for *Agile HW Development* is present to some extent, but still, the majority of the papers leave agile development undefined.

7.2 RQ3: What Methods and Practices Are Proposed to Increase Agility of the SoC Development?

The proceeding analysis is not an overview of the methodologies in general. Instead, the analysis provides a view of the current state of agile HW development. One should be careful when interpreting the results. Although the review shows approaches to gain agility, applying these methods does not directly lead to agile development. For example, an SoC developed by HW/SW co-design methodology can still be applied using conventional, non-agile methods. As we notice already in the mapping study, proposals often introduce the usage of the multiple methods. Due to that, a single publication can be discussed in multiple sub-sections below.

The proposals are grouped into the following categories.

- MDD: Abstraction, Model transformations, code generation and synthesis.
- Development platforms.
- HW-SW co-design.
- Open source hardware.

7.2.1 Abstraction, Model Transformations, Code Generation and Synthesis. Allen et al. [4] propose Rapid HDL based on the Microsoft .NET platform and utilizes .NET Common Language Runtime, which includes support for C#, Vb.Net, Ruby, F#, Cobal, and others.

The Chisel [31] is based on Scala language and thus inherits the modern SW language features such as parameterized types, abstract data types, operator overloading, and type inference. The Chisel generates to RTL HDL model of the HW, but also the Chisel model can be translated to a cycle-accurate C++ executable for fast simulation. The latest versions of Chisel also include **Flexible Internal Representation for RTL (FIRRTL)** as a platform for writing circuit-level transformations. In addition to Reference [31], Chisel has been applied in References [5, 7, 12, 16, 25, 36, 39], among the publications included to this literature review.

Trapaglia et al. [43] presents DUTILS, a python-based HW-SW co-design platform. The main tool is the DUTILS python class library, which abstracts the HW under simulation. It is used together with the CocoTB python class library for HW verification. The benefit is an incremental refinement of the python-based system models toward use-case-specific HW models. The proposal improves the reuse of verification environments and tests between the high-level models and RTL models.

The python-based PyMTL framework is proposed by Jiang et al. [20]. PyMTL3 supports multi-level modeling on functional, cycle-accurate, and register-transfer levels. One of the key design principles is the modularity of the framework. That is achieved by dividing the platform into frontend, intermediate representations and passes similar to the LLVM compiler architecture. The OpenPiton platform [8] uses also a PyMTL-based PyOCN network on chip generator in their work on developing custom SoC chips.

The PyRTL [13] is a python library for HW modeling similar to PyMTL3. The main difference is that it is a compact core library for HW development, which provides simplicity, usability, and clarity. The PyRTL can be extended with standard python modules, such as math modules to extend modeling capabilities. Verilog and Chisel FIRRTL code can be generated from PyRTL.

The Chipkit [46] uses a template-based code generation implemented in python to generate control and status registers for HW implementation with related SW API and documentation.

Mantovani et al. [32] use C/C++/SystemC class templates for different HLS tools. Also, their design tool with a graphical user interface generates tile (sub-system) skeletons based on selected features.

Bahr et al. [6] present an HW-SW compiler framework to generate HW accelerators in coarse-grained reconfigurable array form. The primary input for the flow is a model written with Halide language. Halide is a C++-based Domain Specific Language for image processing applications.

Minutoli et al. [33] propose an LLVM compiler-based multi-layer synthesis framework to generate hardware descriptions of the machine learning accelerators. Multiple front-ends and back-ends can be supported. The framework is demonstrated with Open Neural Network Exchange to Chisel FIRRTL synthesis through multiple optimizations.

Shalf et al. [41] combine model transformation and optimization for scientific applications written for example in C++ or Fortran. They utilize commercial Tensilica Xtensa Processor Generator tool flow to explore and different processor design choices to optimize applications and HW.

The DEC++ by Sorensen et al. [42] is an LLVM-based compiler for model transformations with related MosaicSim simulator to perform HW-SW co-design of the heterogeneous systems. Different target back-ends for multiple processor ISA such as x86 and RISC-V enables architectural exploration of the system. The DEC++ does not produce synthesizable RTL HW models but provides common SW API for custom HW accelerators.

To summarize, high-level programming language–based abstraction, code generation, and synthesis have been used to increase agility of the HW development. Chisel, python, and LLVM compiler frameworks are common technologies among the publications.

7.2.2 Development Platforms. Amid et al. [5] introduces Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs. The key tool is a Chisel-based Rocket Chip RISC-V CPU SoC generator, FPGA-accelerated simulation with FireSim, and modular VLSI flow named Hammer. Hammer provides an abstraction of the process technology and EDA tool-specific concerns, which enables easier changes to the ASIC technology.

The OpenPiton [8] platform consists of a reused Ariane Linux-capable 64-bit RISC-V CPU core, PyOCN network on chip generator compilation framework for open source EDA tools, and FuseSoC IP management tool.

The ESP [32] is a GUI-based tool covering accelerator IPs and SoC design tool flows. The ESP platform addresses verification, FPGA synthesis, device drivers, and test applications based on design parameters defined by the user. On the SoC, level connectivity is also addressed in the form of generated routing tables, memory maps, device trees, and SW header files. The ESP platform has also been used for ASIC design by Bose et al. [11].

The python-based development platforms [13, 20, 43] gain mainly from the ability to develop modeling, digital design, and verification with the same input language and thus provide a unified development platform for developers.

Eschweiler et al. [18] base their platform on top of QEMU and KVM HW virtualization tools that can address HW modeling and SW development.

The DEC++ [42] supports platform thinking through the use of the LLVM framework, which harmonizes different model transformations (LLVM passes).

The incremental synthesis and simulation framework LiveHD [45] is based on the LLVM-like LGraph, which is a unified data model for open source HW design. The work highlights the slow iteration time of the SoC simulation and synthesis tools as a key challenge for agility. In summary, we notice that development platforms are built around modeling with programming languages, integration tools to increase automation, or tool flows. In addition, extending some existing frameworks like LLVM compiler is present. The code generation makes the best sense when multiple different targets are generated. Thus, the code generation and development platforms often exist in combination.

Some platform architectures are present in Reference [32] (ESP) and Reference [35] (Blackparrot), but the border between the development platform and the platform architecture is not always clear. These "platform development platforms" come with a reference architecture as a starting point.

7.2.3 HW-SW Co-design. HW-SW co-design-based approach by Sorensen et al. [42] is based on a compiler that automates architectural exploration of heterogeneous systems. The flow supports popular programming languages such as C++ and python. Trapagliat et al. [43] bases HW-SW co-design support to reuse between the HW design and the verification. Automation is not involved when refining from SW-based models to HW implementations. Instead, they talk about fluent migration from the models to the implementation. The strategy of proposals by Shalf et al. [41] is highly HW-SW co-design focused.

Wagner et al. [44] highlight the importance of HW-SW co-design for agile development but do not present a detailed methodology to implement it.

Despite a relatively small number of publications, the typical HW-SW co-design flows are present. In a summary, the proposals focus more on exploration and high-level models rather than

the development of lower-level implementations. The HLS approach proposed in Reference [32] could be seen as HW/SW co-design, but the authors did not consider it to this category.

7.2.4 Open Source Hardware. The open source HW is present to great extent, but we focused only on contributions that increase development efficiency and agility.

Lee et al. [31] outline free, open, and extensible RISC-V ISA as one of the keys for implementing their agile HW manifesto. Openness together with ISA extensions provides a possibility to explore different architectural changes with small teams. Also, the Chisel language is an open source implementation.

The Hammer for digital design and the BAG for analog and mixed-signal are ASIC technology and tool-agnostic abstraction layers for physical design [5]. These tools help in exchanging flow-related information in an open source manner. Exchanging design information in a commercial physical design tool format is often forbidden.

As the name suggests, OpenPiton [8] takes openness as its key discipline. They use and contribute open source HW and also share important insights to build and maintain such a community.

PyMTL3 [20] is released as open source but, more importantly, from an agility point of view, emphasizes interoperability with other open source hardware tools.

The ESP platform [32] contributes to the open source HW community by providing an integration platform for heterogeneous systems to gain agility on the SoC level.

The Blackparrot [35] base their fast development iterations on the open source RTL component libraries BaseJump STL and HardFloat.

In summary, openness is addressed from multiple perspectives. In many cases, the link between agile development and openness is implied but remains weak. Open source could potentially increase flexibility in the long run by allowing faster and cheaper experiments, easing reuse, and opening completely new possibilities by making SoC design affordable for a wider audience. However, we have identified in our previous work also quality issues in open source HW such as lack of documentation and varying quality [37].

As a whole, when we return to *RQ3* we notice that literature addresses the challenges of SoC development time and provides some solutions to improve it. However, the link to agile development remains weak.

8 SUMMARY AND DISCUSSION OF THE RESULTS

The number of publications and the presence in high-impact forums shows increasing interest in agile SoC HW development. The research seems to be in the early phase and not even agreed definitions exist compared to agile SW development. One explanation might be that HW designers are more artifact oriented and have not yet seen the importance of the development process. However, the long feedback loops and highly dependent activities narrow the space for agility.

We summarize our findings from the mapping study and literature review in the following by addressing each research question:

- RQ1: The papers covered exploration and development lifecycles of the process but hardly anything on process management.
- RQ2: None of the famous SW agile methods were literally applied to the SoC HW development. The agile HW manifesto is the most established set of principles this far.
- RQ3: Model-Driven Development was not explicitly mentioned, but many papers addressed abstract models and code generation to improve productivity and quality. However, quantified results on the development efficiency are completely missing.

- RQ4: Several ASIC implementations were found, which basically fit well to the idea of frequent prototyping in agile development. However, the mindset change from "first time right" is not yet seen and projects follow more traditional methods.

Open standards, open source component libraries, and open source tools are clearly present in the literature. Open source HW will be the game changer. Open source EDA tools already provide cheaper experimenting with promising early results [22]. Commercial tools will stay de facto for production quality designs, but open source helps the development community to freely exchange ideas and experiences. Agility can break out from the new developer communities whose members might not have an HW design background.

9 CONCLUSIONS

This article presented the results of the literature review and the mapping study to reveal the current state of agile SoC HW development. Twenty-five selected papers were included and analyzed based on accurately defined research methodology. The baseline for the mapping study was our own SoC development process description. It follows the conventional approach, but helped us to find the relevant research questions and mapping study attributes. The existing agile methods were introduced, and the research methodology was described in detail for independent reconstruction.

As a summary of the results, little has been published about agile SoC HW development, but there is increasing interest in the topic. Only the agile HW manifesto by Lee et al. [31] can be regarded as a defined agile HW development method, but in general researchers are missing an agreed-on definition for agile HW development. Moreover, the link to agile development methods rooted to SW development was surprisingly weak. Multiple SoC-specific development methodologies and practices were proposed to increase agility of the development, but none of them were novel, and we see them more as enablers toward the agile instead of the agile methods as such.

Agile development is not just a set of development or project practices, and changes in the mindset are needed to become agile to a great extent. None of the contribution provides such a holistic view, but the agile HW manifesto includes the set of principles in addition to proposed tools to help the situation.

To be able to address agile SoC HW development better, there should be more data available about the used methods across the full SoC development span, including project management aspects. The SoC HW research should also focus more on reporting methodological results, especially on large-scale projects such as amount and duration of the development iterations, resourcing, and project schedule. The open source HW and the open source EDA tools could act as as relevant source of data while it could increase transparency of the work in academic and industry.

The results give good insight to the latest advancements in agile SoC HW development. The identified gaps need to be addressed before the SoC development can become agile enough to solve the design challenges in modern complex System-on-Chips. We look forward to seeing more scientific papers reporting on SoC project calendar timelines and development efforts and details of the development methodologies in the future.

APPENDIX

A LITERATURE REVIEW AND MAPPING STUDY DATA

Table 3. The List of Included Literature and the Summary of Mapping Study Results for Publications P1–P12

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
P1, [4]	2009	HW	FPGA	L2	3	NA	Rising abstraction, Code gen. for digital design	S	Prop.
P2, [5]	2020	HW	ASIC, FPGA, Sim	L1, L2	1–7	Agile HW manifesto	Development platform, Open source, Rising abstraction, Code gen.	NA	CSA, ASIC
P3, [7]	2019	HW	ASIC	L1, L2	1–4	Agile HW manifesto	Rising abstraction, Generator-based design, Code gen.	Large	CSA, ASIC
P6, [6]	2020	HW	ASIC	L1, L2	1, 3, 4, 7	NA	Rising abstraction, Code gen.	NA	CSA
P5, [8]	2020	HW	ASIC, FPGA	L1, L2	1, 3, 6	NA	Development platform, Rising abstraction, Code gen.	NA	CSI, ASIC
P6, [11]	2021	Sys.	ASIC, FPGA	L1, L2	3, 7	NA	Development platform (ESP) [32]	Large	CSM, ASIC
P7, [12]	2021	HW	FPGA	L2	3–5	Agile HW manifesto	NA	NA	CSI, FPGA
P8, [13]	2020	HW	Sim	L1, L2	1, 3–5	TDD	Development platform, Rising abstraction, Code gen.	NA	Prop.
P9, [16]	2021	HW	Sim	L1, L2	1, 3, 5	Agile HW manifesto	Rising abstraction (Chisel verification improvement)	NA	CSA
P10, [18]	2015	SW	Sim	L2	1, 6, 7	TDD	Rapid prototyping, Development platform	S	Prop., CSA
P11, [20]	2020	HW	Sim	L1, L2	1, 3, 4	Agile HW manifesto, TDD	Development platform, Rising abstraction, Code gen., open source tool interoperability	NA	Prop.
P12, [25]	2021	HW	ASIC	L1, L2	1,3	Agile HW manifesto	NA	NA	CSM, ASIC

Table 4. The List of Included Literature and the Summary of Mapping Study Results for Publications P13-P25

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
P13, [29]	2016	Sys, SW	ES, IC	L2, PM	NA	SCRUM	NA	S and L	CSI
P14, [31]	2016	HW	ASIC, FPGA, Sim	L2, PM	1, 3–6	Agile HW manifesto	Rising abstraction, Generator-based design, Open architecture, continuous and iterative digital design	S	CSA, ASIC
P15, [32]	2020	HW	FPGA	L2	3, 6, 7	NA	Platform architecture, Open source components, High-level Synthesis, templates and code gen.	NA	CSA, FPGA
P16, [33]	2020	HW	ASIC	L1, L2	1, 3, 4	NA	Multi-layer synthesis flow from ML-models to HW description, rising abstraction	NA	CSA
P17, [35]	2020	HW	ASIC	L1, L2	1, 3–5	Principles Prop.: Be tiny, be modular, be friendly (approachable)	Rising abstraction, Platform architecture, Open source components, Evaluating design complexity and approachability	NA	CSA, ASIC
[12] P18, [36]	2021	HW	FPGA	L1, L2	1, 3–5	Agile HW manifesto	NA	NA	CSA, FPGA
P19, [39]	2021	HW	ASIC	L1, L2	1–5	Agile HW manifesto	Code generation, Physical design tool abstraction (Hammer)	S.	CSA, ASIC
P20, [41]	2011	HW	FPGA, Sim	L1, L2	1, 3, 6, 7	NA	HW-SW co-design, Modeling, Model transformation, Code gen., HW emulation.	NA	CSA
P21, [42]	2020	HW	Sim	L1	1	NA	HW-SW co-design, Development platform, Modeling of heterogeneous system, LLVM compiler architecture.	NA	CSA
P22, [43]	2019	HW	FPGA, Sim	L1, L2	1, 5, 6	NA	HW-SW co-design, python-based development platform, Abstracting RTL Sim.	NA	CSA
P23, [44]	2014	SW	ES, Sim	PM	1, 3, 7	SCRUM, Agile Release Train	HW-SW co-design, Sim models of the HW.	NA	Prop.
P24, [45]	2020	HW	ASIC, FPGA, Sim	L2	3–5	NA	Development platform for open source EDA tools, Incremental digital design, intermediate model for tool coupling.	NA	CSA
P25, [46]	2020	HW	ASIC	L2	3–6	NA	Code gen., SystemVerilog coding guidelines, Wrapping of ASIC technology models	NA	CSI

REFERENCES

[1] Pekka Abrahamsson, Kieran Conboy, and Xiaofeng Wang. 2009. 'Lots done, more to do': The current state of agile systems development research. 281–284.

[2] Pekka Abrahamsson, Nilay Oza, and Mikko T. Siponen. 2010. Agile software development methods: A comparative review. In *Agile Software Development*. Springer, 31–59.

[3] Muhammad Ovais Ahmad. 2019. Agile methods and cyber-physical systems development-A review with preliminary analysis. In *International Conference on Big Data and Security*. Springer, 274–285.

[4] Jacob N. Allen, Hoda S. Abdel-Aty-Zohdy, and Robert L. Ewing. 2009. Agile hardware development with rapid hardware definition language. In *Proceedings of the IEEE International Conference on Electro/Information Technology*. IEEE, 383–388.

[5] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. 2020. Chipyard: Integrated design, simulation, and implementation framework for custom SoCs. *IEEE Micro* 40, 4 (2020), 10–21.

[6] Rick Bahr, Clark Barrett, Nikhil Bhagdikar, Alex Carsello, Ross Daly, Caleb Donovick, David Durst, Kayvon Fatahalian, Kathleen Feng, Pat Hanrahan, et al. 2020. Creating an agile hardware design flow. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC'20)*. IEEE, 1–6.

[7] Steven Bailey, Paul Rigge, Jaeduk Han, Richard Lin, Eric Y. Chang, Howard Mao, Zhongkai Wang, Chick Markley, Adam M. Izraelevitz, Angie Wang, et al. 2019. A mixed-signal RISC-V signal analysis SoC generator with a 16-nm FinFET instance. *IEEE J. Solid-State Circ.* 54, 10 (2019), 2786–2801.

[8] Jonathan Balkind, Ting-Jung Chang, Paul J. Jackson, Georgios Tziantzioulis, Ang Li, Fei Gao, Alexey Lavrov, Grigory Chirkov, Jinzheng Tu, Mohammad Shahrad, et al. 2020. OpenPiton at 5: A nexus for open and agile hardware design. *IEEE Micro* 40, 4 (2020), 22–31.

[9] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. 2001. The agile manifesto. https://agilemanifesto.org/. Accessed: 2022-07-31.

[10] Barry W. Boehm. 1988. A spiral model of software development and enhancement. *Computer* 21, 5 (1988), 61–72.

[11] Pradip Bose, Augusto Vega, Sarita Adve, Vikram Adve, Sasa Misailovic, Luca Carloni, Ken Shepard, David Brooks, Vijay Janapa Reddi, and Gu-Yeon Wei. 2021. Secure and resilient SoCs for autonomous vehicles. In *Proceedings of the 3rd International Workshop on Domain Specific System Architecture (DOSSA'21)*. 1–6.

[12] Lucas Cordeiro, Carlos Mar, Eduardo Valentin, Fabiano Cruz, Daniel Patrick, Raimundo Barreto, and Vicente Lucena. 2008. A platform-based software design methodology for embedded control systems: An agile toolkit. In *Proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS'08)*. IEEE, 408–417.

[13] Deeksha Dangwal, Georgios Tzimpragos, and Timothy Sherwood. 2020. Agile hardware development and instrumentation with PyRTL. *IEEE Micro* 40, 4 (2020), 76–84.

[14] Surafel Demissie, Frank Keenan, Özden Özcan-Top, and Fergal McCaffery. 2018. Agile usage in embedded software development in safety critical domain—A systematic review. In *International Conference on Software Process Improvement and Capability Determination*. Springer, 316–326.

[15] Kim Dikert, Maria Paasivaara, and Casper Lassenius. 2016. Challenges and success factors for large-scale agile transformations: A systematic literature review. *J. Syst. Softw.* 119 (2016), 87–108.

[16] Andrew Dobis, Tjark Petersen, Hans Jakob Damsgaard, Kasper Juul Hesse Rasmussen, Enrico Tolotto, Simon Thye Andersen, Richard Lin, and Martin Schoeberl. 2021. Chiselverify: An open-source hardware verification library for chisel and scala. In *Proceedings of the IEEE Nordic Circuits and Systems Conference (NORCAS'21)*. IEEE, 1–7.

[17] Tore Dybå and Torgeir Dingsøyr. 2008. Empirical studies of agile software development: A systematic review. *Inf. Softw. Technol.* 50, 9-10 (2008), 833–859.

[18] Dominic Eschweiler and Volker Lindenstruth. 2015. Test driven development for device drivers and rapid hardware prototyping. In *Proceedings of the IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW'15)*. IEEE, 1–9.

[19] Daniel D. Gajski and Robert H. Kuhn. 1983. New VLSI tools. *Computer* 16, 12 (1983), 11–14.

[20] Shunning Jiang, Peitian Pan, Yanghui Ou, and Christopher Batten. 2020. PyMTL3: A python framework for open-source hardware modeling, generation, simulation, and verification. *IEEE Micro* 40, 4 (2020), 58–66.

[21] Lizy Kurian John. 2020. Agile hardware design. *IEEE Ann. Hist. Comput.* 40, 04 (2020), 4–5.

[22] Andrew B. Kahng and Tom Spyrou. 2021. The OpenROAD project: Unleashing hardware innovation. In *Proceedings of the Government Microcircuit Applications and Critical Technology Conference*. 1–6.

[23] Matti Kaisti, Ville Rantala, Tapio Mujunen, Sami Hyrynsalmi, Kaisa Könnölä, Tuomas Mäkilä, and Teijo Lehtonen. 2013. Agile methods for embedded systems development-a literature review and a mapping study. *EURASIP J. Embed. Syst.* 2013, 1 (2013), 15.

[24] Antti Kamppi, Esko Pekkarinen, Janne Virtanen, Joni-Matti Määttä, Juho Järvinen, Lauri Matilainen, Mikko Teuho, and Timo D. Hämäläinen. 2017. Kactus2: A graphical EDA tool built on the IP-XACT standard. *J. Open Source Softw.* 2, 13 (2017), 151.

[25] Sagar Karandikar, Chris Leary, Chris Kennelly, Jerry Zhao, Dinesh Parimi, Borivoje Nikolic, Krste Asanovic, and Parthasarathy Ranganathan. 2021. A hardware accelerator for protocol buffers. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-54)*. 462–478.

[26] Michael Keating and Pierre Bricaud. 2002. *Reuse Methodology Manual for System-on-a-Chip Designs: For System-on-a-chip Designs*. Springer Science & Business Media.

[27] Staffs Keele et al. 2007. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Technical Report. Citeseer.

[28] Kurt Keutzer, A. Richard Newton, Jan M. Rabaey, and Alberto Sangiovanni-Vincentelli. 2000. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. Comput.-aid. Des. Integr. Circ. Syst.* 19, 12 (2000), 1523–1543.

[29] Kaisa Könnölä, Samuli Suomi, Tuomas Mäkilä, Tero Jokela, Ville Rantala, and Teijo Lehtonen. 2016. Agile methods in embedded system development: Multiple-case study of three industrial cases. *J. Syst. Softw.* 118 (2016), 134–150.

[30] Maarit Laanti, Jouni Similä, and Pekka Abrahamsson. 2013. Definitions of agile software development and agility. In *European Conference on Software Process Improvement*. Springer, 247–258.

[31] Yunsup Lee, Andrew Waterman, Henry Cook, Brian Zimmer, Ben Keller, Alberto Puggelli, Jaehwa Kwak, Ruzica Jevtic, Stevo Bailey, Milovan Blagojevic, et al. 2016. An agile approach to building RISC-V microprocessors. *IEEE Micro* 36, 2 (2016), 8–20.

[32] Paolo Mantovani, Davide Giri, Giuseppe Di Guglielmo, Luca Piccolboni, Joseph Zuckerman, Emilio G. Cota, Michele Petracca, Christian Pilato, and Luca P. Carloni. 2020. Agile SoC development with open ESP. In *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD'20)*. IEEE, 1–9.

[33] Marco Minutoli, Vito Giovanni Castellana, Cheng Tan, Joseph Manzano, Vinay Amatya, Antonino Tumeo, David Brooks, and Gu-Yeon Wei. 2020. SODA: A new synthesis infrastructure for agile hardware design of machine learning accelerators. In *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD'20)*. IEEE, 1–7.

[34] Maria Paasivaara. 2017. Adopting SAFe to scale agile in a globally distributed organization. In *Proceedings of the IEEE 12th International Conference on Global Software Engineering (ICGSE'17)*. IEEE, 36–40.

[35] Daniel Petrisko, Farzam Gilani, Mark Wyse, Dai Cheol Jung, Scott Davidson, Paul Gao, Chun Zhao, Zahra Azad, Sadullah Canakci, Bandhav Veluri, et al. 2020. BlackParrot: An agile open-source RISC-V multicore for accelerator SoCs. *IEEE Micro* 40, 4 (2020), 93–102.

[36] M. L. Petrović and V. M. Milovanović. 2021. A chisel generator of parameterizable and runtime reconfigurable linear insertion streaming sorters. In *Proceedings of the IEEE 32nd International Conference on Microelectronics (MIEL'21)*. IEEE, 251–254.

[37] Antti Rautakoura, Timo Hämäläinen, Ari Kulmala, Tero Lehtinen, Mehdi Duman, and Mohamed Ibrahim. 2022. Ballast: Implementation of a large MP-SoC on 22nm ASIC technology. In *Proceedings of the 23rd Euromicro Conference on Digital System Design (DSD'22)*. IEEE, 276–283.

[38] Antti Rautakoura, Matti Käyrä, Timo D. Hämäläinen, Wolfgang Ecker, Esko Pekkarinen, and Mikko Teuho. 2020. Kamel: IP-XACT compatible intermediate meta-model for IP generation. In *Proceedings of the 25th Euromicro Conference on Digital System Design (DSD'20)*. IEEE, 325–331.

[39] Colin Schmidt, John Wright, Zhongkai Wang, Eric Chang, Albert Ou, Woorham Bae, Sean Huang, Vladimir Milovanović, Anita Flynn, Brian Richards, et al. 2021. An eight-core 1.44-GHz RISC-V vector processor in 16-nm FinFET. *IEEE J. Solid-State Circ.* 57, 1 (2021), 140–152.

[40] Julian Immanuel Schrof, Alexander Atzberger, Efthymios Papoutsis, and Kristin Paetzold. 2019. Potential of technological enablement for agile automotive product development. In *Proceedings of the IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC'19)*. IEEE, 1–8.

[41] John Shalf, Dan Quinlan, and Curtis Janssen. 2011. Rethinking hardware-software codesign for exascale systems. *Computer* 44, 11 (2011), 22–30.

[42] Tyler Sorensen, Aninda Manocha, Esin Tureci, Marcelo Orenes-Vera, Juan L. Aragón, and Margaret Martonosi. 2020. A simulator and compiler framework for agile hardware-software co-design evaluation and exploration. In *Proceedings of the IEEE/ACM International Conference On Computer Aided Design (ICCAD'20)*. IEEE, 1–9.

[43] Matias Trapaglia, Ricardo Cayssials, Lorenzo De Pasquale, and Edgardo Ferro. 2019. Flexible software to hardware migration methodology for FPGA design and verification. In *Proceedings of the X Southern Conference on Programmable Logic (SPL'19)*. IEEE, 39–44.

[44] Stefan Wagner. 2014. Scrum for cyber-physical systems: A process proposal. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*. ACM, 51–56.

[45] Sheng-Hong Wang, Rafael Trapani Possignolo, Haven Blake Skinner, and Jose Renau. 2020. LiveHD: A productive live hardware development flow. *IEEE Micro* 40, 4 (2020), 67–75.

[46] Paul N. Whatmough, Marco Donato, Glenn G. Ko, Sae Kyu Lee, David Brooks, and Gu-Yeon Wei. 2020. CHIPKIT: An agile, reusable open-source framework for rapid test chip development. *IEEE Micro* 40, 4 (2020), 32–40.

Received 19 August 2022; revised 11 November 2022; accepted 30 November 2022