1.3

Foundational Technology Layers

EMBEDDED SOFTWARE AND BEYOND

1.3.1. Introduction

The Artemis/Advancy report [1] states that "the investments in software technologies should be on at least an equal footing with hardware technologies, considering the expected growth at the higher level of the value chain (Systems of Systems, applications and solutions)". According to the same report, embedded software[2] technology and software engineering tools are part of the six technology domains for embedded intelligence. Embedded intelligence is the ability of a system or component to reflect on its own state (e.g. operational performance, usage load, environment), and as such is a necessary step towards the level of digitalisation and sustainability that is aimed for. In this context, embedded intelligence supports the green deal initiative, as one of the tools for reaching sustainability.
Embedded software enables embedded and cyber-physical systems (ECPS) in a way that they can play a key role in solutions for digitalisation in almost every application domain (cf. Chapters 3.1-3.6). The reason for this Chapter being entitled **Embedded Software and Beyond** is to stress that embedded software is a key component of system's internal intelligence, it enables systems to act on external events, and it enables inter-system communication.

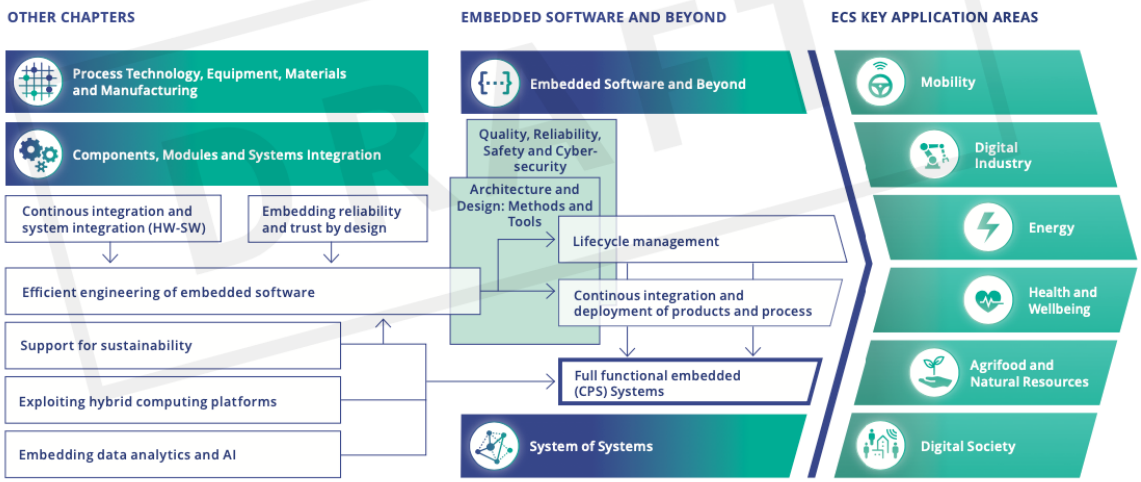

Figure 1 Positioning of the Embedded Software and Beyond Chapter in the ECS-SRIA

Figure 1 illustrates the role and positioning of the **Embedded Software and Beyond** Chapter in the ECS- SRIA. The Chapter on **Components Modules and System Integration** is focused on functional hardware components and systems that compose the embedded and cyber-physical systems (CPS), considered in this Chapter. While the System of Systems (SoS) Chapter is based on independent, fully functional systems, products and services (which are also discussed in this Chapter), they are also the constituents of SoS solutions. The **Architecture and Design: Methods and Tools** Chapter examines engineering processes, methods, and tools, while this Chapter focuses more on the engineering aspects of **Embedded Software**

[1] Advancy, 2019: Embedded Intelligence: Trends and Challenges, A study by Advancy, commissioned by ARTEMIS Industry Association. March 2019. Downloadable from: https://www.inside-association.eu/publications
[2] As defined in <<>> https://docs.inside-association.eu/index.php/apps/richdocuments/index?fileId=9080&dir=%2FECS-SRIA%2F2023%2F0%20Introduction%20and%20overview%2F001.draft%20version

and Beyond. For the discussion of safe, trustworthy, and explainable AI in the context of embedded intelligence this Chapter is also linked to the **Quality, Reliability, Safety and Cybersecurity** Chapter <<and to the "EdgeAI" Chapter 2.1 >>.

From a functional perspective, the role of Embedded and Cyber-Physical Systems (ECPS) in complex systems is becoming increasingly dominant (e.g. cars, trains, airplanes and health equipment) because of the new software-enabled functionalities they provide (including aspects as security, privacy and autonomy). In these systems, most of the innovations nowadays come from software. ECPS are also required for the interconnection and interoperability of systems in SoS (e.g. smart cities, air traffic management). Owing to all these factors, ECPS are an irreplaceable part of the strive towards digitalisation of our society.
At the same time, ECPS need to exhibit required quality properties (e.g. safety, security, reliability, dependability, sustainability, and, ultimately, trustworthiness). Furthermore, due to their close integration with the physical world, ECPS must take into account the dynamic and evolving aspects of their environment to provide deterministic, high-performance, and low-power computing, especially when processing intelligent algorithms. Increasingly, software applications will run as services on distributed SoS involving heterogeneous devices (e.g: servers, edge devices) and networks, with a diversity of resource restrictions. In addition, it is required from ECPS that its functionalities and its hardware capabilities evolve and adapt during their lifecycle – e.g. through updates of software or hardware in the field and/or by learning. Building these systems and guaranteeing their previously mentioned quality properties, along with supporting their long lifetime and certification, requires innovative technologies in the areas of modelling, software engineering, model-based design, verification and validation (V&V) technologies, and virtual engineering. These advances need to enable engineering of high-quality, certifiable ECPS that can be produced (cost-)effectively (cf. Chapter 2.3, **Architecture and Design: Methods and Tools**).

DRAFT

1.3.2. Scope

The scope of the challenges existing in embedded software engineering for ECPS includes:

- Interoperability.
- Complexity.
- Software quality (safety, security, performance prediction and run-time performance, reliability, dependability, sustainability, and, ultimately, trustworthiness).
- Lifecycle (maintainability, extendibility).
- Efficiency, effectiveness, and sustainability of software development.
- Dynamic environment of ECPS and adaptability.
- Maintenance, integration, rejuvenation and extendibility of legacy software solutions.

To enable ECPS functionalities and their required level of interoperability, the engineering process will be progressively automated and will need to be integrated in advanced SoS engineering covering the whole product during its lifetime. Besides enabling new functionalities and their interoperability, it will need to cover non-functional requirements (safety, security, run-time performance, reliability, dependability, sustainability, and, ultimately, trustworthiness) visible to end users of ECPS, and to also satisfy quality

requirements important to engineers of the systems (e.g. evolution, maintenance). This requires innovative technologies that can be adapted to the specific requirements of ECPS and, subsequently, SoS.

Further complexity will be imposed by the introduction of Artificial Intelligence (AI), machine-to-machine (M2M) interaction, new business models, and monetisation at the edge. This provides opportunities for enhancing new engineering techniques like AI for SW engineering, and SW engineering for AI. Future software solutions in ECPS will solely depend on new software engineering tools and engineering processes (e.g., quality assurance, Verification and Validation (V&V) techniques and methods on all levels of individual IoT and in the SoS domain).

Producing industrial software, and embedded software in particular, is not merely a matter of writing code: to be of sufficient quality, it also requires a strong scientific foundation to assure correct behavior under all circumstances. Modern software used in products such as cars, airplanes, robots, banks, healthcare systems, and the public services comprises millions of lines of code. To produce this type of software, many challenges have to be overcome. Even though software in ECPS impacts everyone everywhere, the effort required to make it reliable, maintainable and usable for longer periods is routinely underestimated. As a result, every day there are news articles about expensive software bugs and over budget or failed software development projects. Currently, there is no clearly reproducible way to develop such software solutions and simultaneously manage their complexity. Also, there exist big challenges with correctness and quality properties of software, as human wellbeing, economic prosperity, and the environment depend on it. There is a need to guarantee that software is maintainable and usable for decades to come, and there is a need to construct it efficiently, effectively and sustainably. Difficulties further increase when legacy systems are considered: information and communications technology (ICT) systems contain crucial legacy components at least 30 years old, which makes maintenance difficult, expensive, and sometimes even impossible.

The scope of this Chapter is research that facilitates engineering of ECPS, enabling digitalisation through the feasible and economically accountable building of SoS with necessary quality. It considers:

- challenges that arise as new applications of ECPS emerge.
- continuous integration, delivery and deployment of products and processes.
- engineering and management of ECPS during their entire lifecycle, including sustainability requirements..

DRAFT

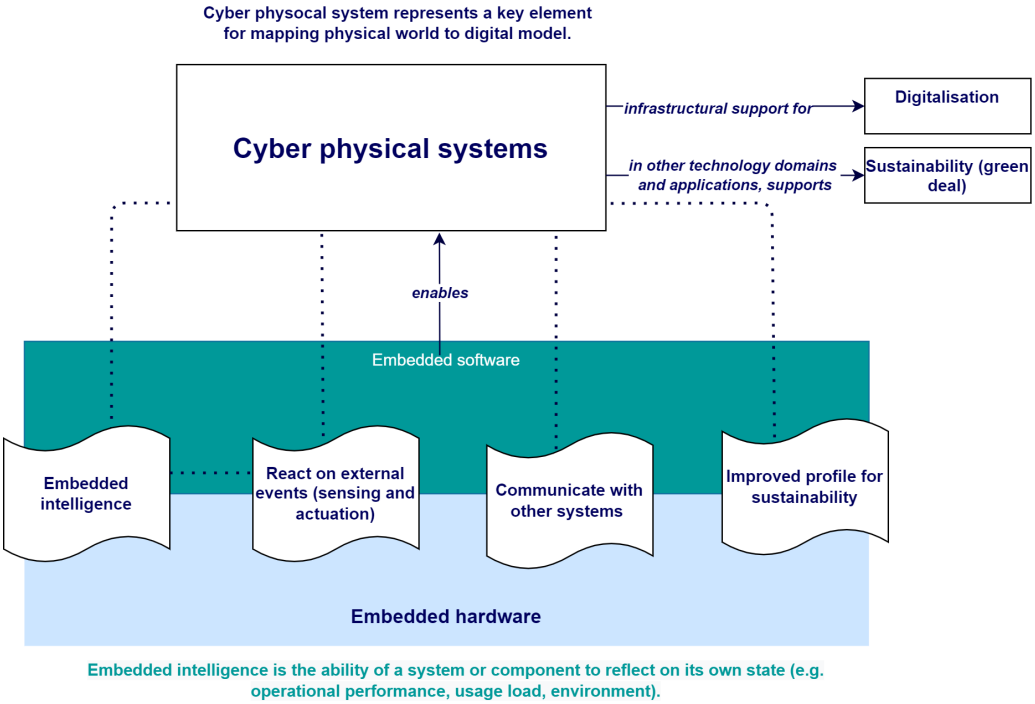

Figure 2 Importance of Embedded Software for Cyber physical systems and its roles.

DRAFT

1.3.3. TECHNOLOGY-ENABLED SOCIETAL BENEFITS

Computing systems are increasingly pervasive and embedded in almost all objects we use in our daily lives. These systems are often connected to (inter)networks, making them part of SoS. ECPS bring intelligence everywhere, allowing data processing and intelligence on the site/edge, improving security and privacy and, through digitalisation, completely changing the way we manage business and everyday activities in almost every application domain (cf. Chapters 3.1-3.6). ECPS also play a critical role in modern digitalisation solutions, quickly becoming nodes in distributed infrastructures supporting SoS for monitoring, controlling and orchestrating of supply chains, manufacturing lines, organisation's internal processes, marketing and sales, consumer products.

Considering their role in digitalisation solutions, ECPS represent a key technology to ensure the continuity of any kind of digital industrial and societal activity, especially during crises, and have an indirect but significant impact on the resilience of economic systems. Without ECPS, data would not be collected, processed, shared, secured/protected, transmitted for further analysis. Embedded software allows for the practical implementation of a large set of such activities, providing the features required by the applications covered in this SRIA, where it becomes a technology enabler. The efficiency and flexibility of embedded software, in conjunction with the hardware capabilities of the ECPS, allows for embedded intelligence on the edge (edge AI), opening unprecedented opportunities for many applications that currently rely on the human acting involvement (e.g. automated driving, security and surveillance, process monitoring). Moreover, digitalisation platforms exploit embedded software flexibility and ECPS features to automate their remote management and control through continuous engineering across their entire lifecycle (e.g. provisioning, bugs

identification, firmware and software updates, configuration management). It is the requirement of embedded software to improve sustainability of these platforms.

1.3.3.1. OPEN SOURCE SOFTWARE AND LICENSES

Free software (FS) is defined by 4 freedoms: the freedom to run as you wish, to study and change the source code, to redistribute copies and to distribute copies of your modified versions[3]. Open source doesn't just mean access to the source code. The Open Source Initiative (OSI) details the distribution terms of open-source software must to comply with the following criteria 10 criteria[4]. Today more than 100 open sources licenses[5] are compliant with these criteria knowing OSI recommends 10 of them because they are popular, widely used, or have strong communities[6]: Apache 2.0, BSD-2 & 3, GPL, LGPL, MIT, Mozilla 2.0, CDDL and EPL 2.0.

Because Open Source components are usually the core building blocks of application software in most innovative domains[7], providing developers with an ever-growing selection of off-the-shelf possibilities that they can use for assembling their products faster and more efficiently, it is essential to understand the benefits and the constrains that come with open source licenses.

The following license spectrum diagram can summarize the freedom from a user point view:

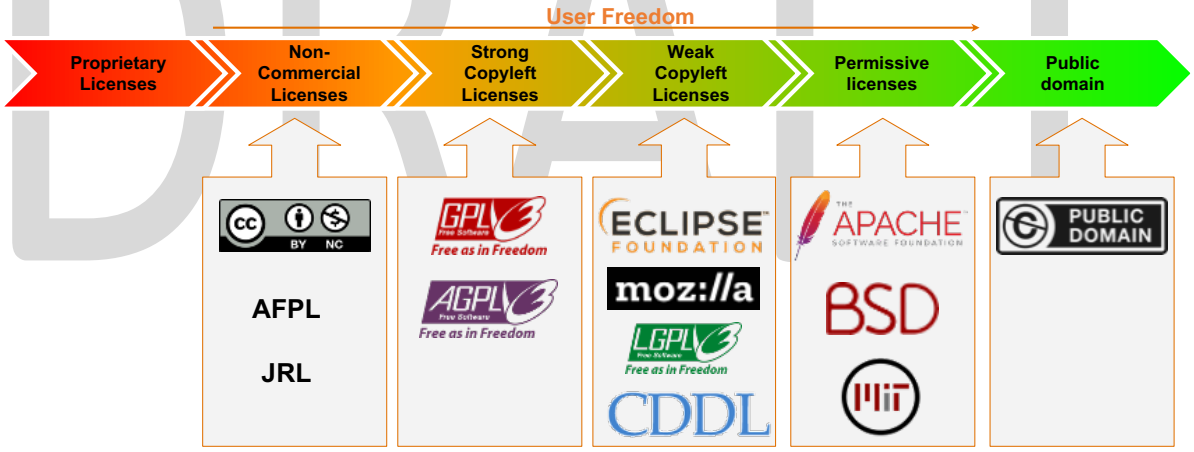

Figure 3: Open License Spectrum

A "strong copyleft" license requires that other code that is used for adding, enhancing, and/or modifying the original work also must inherit all the original work's license requirements such as to make the code publicly available. The most notably strong copyleft licenses are GPL, and AGPL. A Weak copyleft license only requires that the source code of the original or modified work is made publicly available, other code that is used together with the work does not necessarily inherit the original work's license requirements. The most known and used copyleft licenses are LGPL and EPL 2.0. A permissive license, instead of copyleft protections, carries only minimal restrictions on how the software can be used, modified, and

[3] https://en.wikipedia.org/wiki/The_Free_Software_Definition
[4] https://opensource.org/docs/definition.php
[5] https://opensource.org/licenses/alphabetical
[6] https://opensource.org/licenses
[7] More than 81% of produced software are consuming open source code in products or services (https://github.com/todogroup/osposurvey/tree/main/2020)

redistributed, usually including a warranty disclaimer. Apache 2.0, BSD and MIT are the most known and used permissive licenses.
<< where is OpenSource advised? Where is closed source/closed IP advised? --> Philippe >>

Throughout the years, the Open Source Software (OSS) development model has gained more and more popularity around the globe. Nowadays, Open Source components are the core building blocks of application software in most innovative domains, providing developers with an ever-growing selection of off-the-shelf possibilities that they can use for assembling their products faster and more efficiently.
Open Source allows to use (freely), study (in complete transparency), improve and share (collaboratively) developed code. To achieve this, Open Source licenses exist that are compliant with the OS definition, which allow the software to be freely used, modified and shared.

2.3.3.1. Application breakthroughs

Embedded software significantly improves the functionalities, features, and capabilities of ECPS, increasing their autonomy and efficiency, and exploiting their resources and computational power, as well as bringing to the field functionalities that used to be reserved only for data centres, or more powerful and resource-rich computing systems. Moreover, implementing specific functionalities in software allows for their re-use in different embedded applications due to software portability across different hardware platforms. Example of increasing computational power of ECPS are video conferencing solutions: less than 20 years ago specialised hardware was still required to realise this function, with big screens in a dedicated set-up that could not be used for any other but a dedicated application. Today, video conferencing is available on every laptop and mobile phone, where the main functionality is implemented by software running on standard hardware. The evolution is pushing to the "edge" specific video conferencing functionalities, adopting dedicated and miniaturised hardware supported by embedded software (video, microphone, and speakers), thus allowing the ECS value chain to acquire a new business opportunity.

Following a similar approach, it has been possible to extend the functionalities of mobile phones and smart watches, which today can a.o. count steps, keep track of walked routes, monitor health, inform users about nearby restaurants, all based on a few extra hardware sensors and a myriad of embedded software applications. The trend is to replace specialised hardware application with software running on generic computing hardware and supported by application-specific hardware, such as AI accelerators, neural chips. This trend is also contributing to the differentiation of the value creation downstream and upstream, as observed in the Advancy report [8](see Figure 4).

These innovations require the following breakthroughs in the field of embedded software:

- Increased engineering efficiency and an effective product innovation process (cf. Chapter 2.3 Architecture and Design: Methods and Tools).

[8] Advancy, 2019: Embedded Intelligence: Trends and Challenges, A study by Advancy, commissioned by ARTEMIS Industry Association. March 2019. Downloadable from: https://www.inside-association.eu/publications

DRAFT

- Enabled adaptable systems by adaptable embedded software and machine reasoning
- Improved system integration and verification and validation.
- Embedded software, and embedded data analytics and AI, to enable system health monitoring, diagnostics, preventive maintenance, and sustainability.
- Data privacy and data integrity.
- Model-based embedded software engineering and design as the basis for managing complexity in SoS (for the latter, cf. Chapter 2.3 Architecture and Design: Methods and Tools).
- Improved multidisciplinary embedded software engineering and software architecting/design for (systems) qualities, including reliability, trust, safety, security, overall system performance, installable, diagnosable, sustainability, re-use (for the latter, cf. Chapter 2.3 Architecture and Design: Methods and Tools and Chapter 2.4 Quality, reliability, safety and cybersecurity).
- Upgradability, dealing with variability, extending lifecycle and sustainable operation

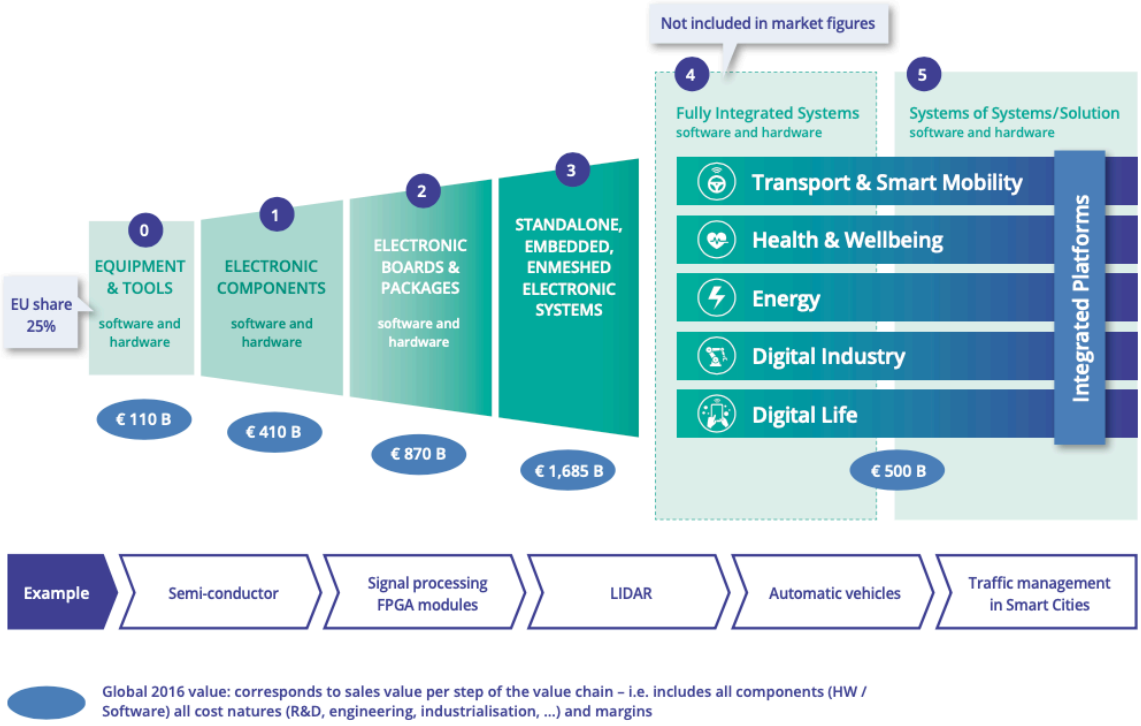

Figure 4 Advancy (2019) [9]report: value creation

1.3.4. STRATEGIC ADVANTAGE FOR THE EU

The ambition of growing competences by researching pervasive embedded software in almost all devices and equipment is to strengthen the digitalisation advance in the EU and the

[9] Advancy, 2019: Embedded Intelligence: Trends and Challenges, A study by Advancy, commissioned by ARTEMIS Industry Association. March 2019. Downloadable from: https://www.inside-association.eu/publications

European position in embedded intelligence and ECPS, ensuring the achievement of world-class leadership in this area through the creation of an ecosystem that supports innovation, stimulates the implementation of the latest achievements of cyber-physical and embedded systems on a European scale, and avoids the fragmentation of investments in research and development and innovation (R&D&I).

European industry that is focused on ECS applications spends about 20% of its R&D efforts in the domain of embedded digital technologies, resulting in a cumulative total R&D&I investment of €150 billion for the period 2013–20. The trend in product and solutions perspective estimates a growth from €500 billion to €3.100–11.100 billion**Error! Bookmark not defined.**, which will be greatly determined by embedded software (30%).

About 60% of all product features will depend on embedded digital technologies, with an estimated impact on the European employment of about 800,000 jobs in the application industries directly resulting from its development.

The current employment levels in the embedded intelligence market in Europe is estimated to be 9.1 million, of which 1.1 million are jobs in the embedded software area, with €15 billion being expected to be allocated to collaborative European R&D&I projects in embedded software and beyond technologies.

DRAFT

1.3.5. MAJOR CHALLENGES

Research and innovation in the domain of embedded software and beyond will have to face six challenges, each generated by the necessity for engineering automation across the entire lifecycle of sustainability, embedded intelligence and trust in embedded software.

- **Major Challenge 1**: efficient engineering of embedded software.
- **Major Challenge 2**: continuous integration and deployment.
- **Major Challenge 3**: lifecycle management.
- **Major Challenge 4**: embedding data analytics and artificial intelligence.
- **Major Challenge 5**: support for Sustainability by embedded software.
- **Major Challenge 6**: software reliability and trust.

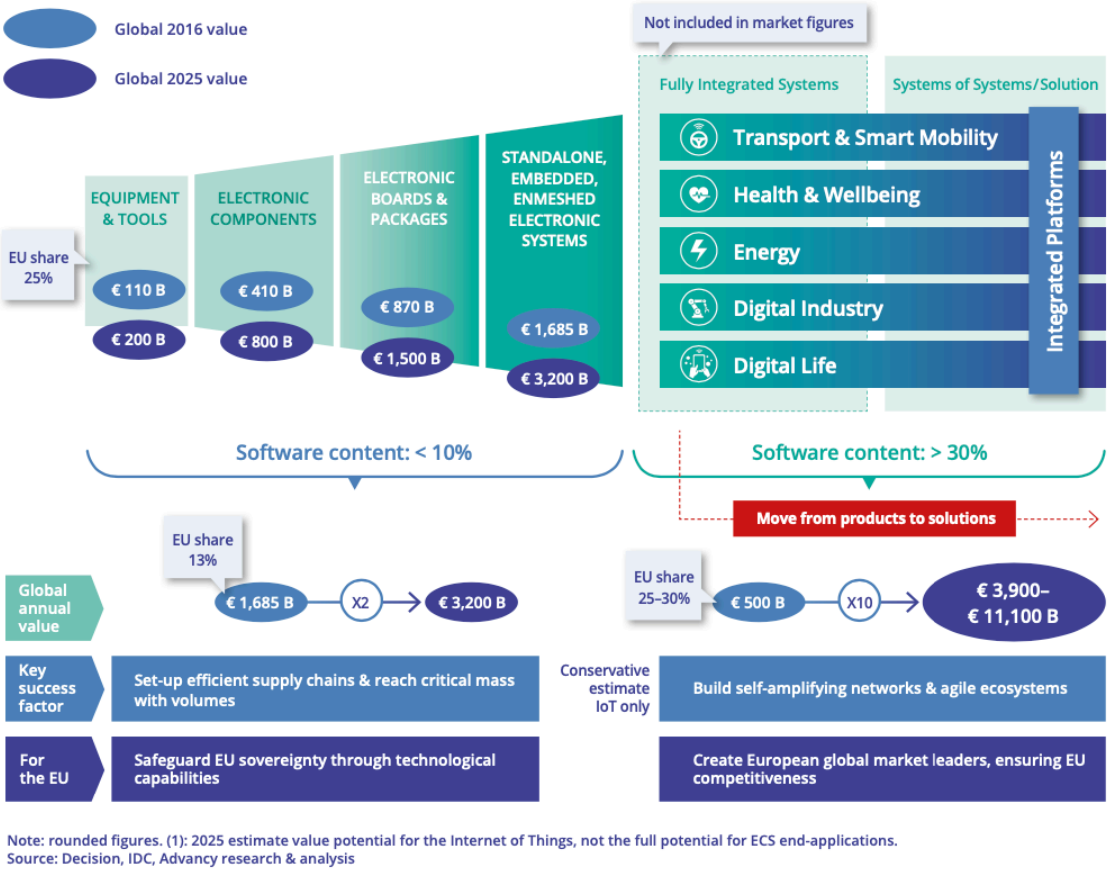

Figure 5 Global and European value chain 2016–25 (Source: Embedded Intelligence: Trends and Challenges, A Study by Advancy[10], Commissioned by ARTEMIS Industry Association, March 2019).

1.3.5.1. Major Challenge 1: efficient engineering of embedded software

1.3.5.1.1. State of the art

Embedded software engineering is frequently more a craft than an engineering discipline, which results in inefficient ways of developing embedded software. This is visible, for instance, in the time required for the integration, verification, validation and releasing of embedded software, which is estimated to exceed 50% of the total R&D&I expenses**Error! Bookmark not defined.**.

A new set of challenges to engineering embedded software is introduced with the emergence of heterogeneous computing architectures into the mainstream. It will be common for embedded systems to combine several types of accelerators to meet power consumption, performance requirements, safety, and real-time requirements. Development, optimisation, and deployment of software for these computing architectures proves to be challenge. If no

[10] Advancy, 2019: Embedded Intelligence: Trends and Challenges, A study by Advancy, commissioned by ARTEMIS Industry Association. March 2019. Downloadable from: https://www.inside-association.eu/publications

solutions that automatically tailor software to specific accelerators [11] [12] are introduced, then developers will be overwhelmed with such effort.

Software engineering is exceeding the human scale, meaning it can no longer be overseen by a human without supporting tools, in terms of velocity of evolution, and the volume of software to be designed, developed and maintained, as well as its variety and uncertainty of conrtext. Engineers require methods and tools to work smarter, not harder, and need engineering process automation and tools and methods for continuous lifecycle support. To achieve these objectives, we need to address the following practical research challenges: shorter development feedback loops; improved tool-supported software development; empirical and automated software engineering; and safe, secure and dependable software platform ecosystems.

1.3.5.1.2. Vision and expected outcome

The demand of embedded software is higher than we can humanly address and deliver, exceeding human scale in terms of evolution speed, volume and variety, as well as in managing complexity. The field of embedded software engineering needs to mature and evolve to address these challenges and satisfy market requirements. In this regard, the following four key aspects must be considered.

(A) From embedded software engineering to cyber physical systems engineering

Developing any high-tech system is, by its very nature, a multi-disciplinary project. There is a whole ecosystem of models (e.g. physical, mechanical, structural, (embedded) software and behavioural) describing various aspects of a system. While many innovations have been achieved in each of the disciplines separately, the entirety still works in silos, each with their own models and tools, and only interfacing at the borders between them. This traditional separation between the hardware and software worlds, and individual disciplines, is hampering the development of new products and services.

Instead of focusing just on the efficiency of embedded software engineering, we already see that the field is evolving into direction of cyber physical systems (cf. Chapter 2.3 **Architecture and Design: Methods and Tools**), and software is one element of engineering. Rather than silos and handovers at the discipline's borders, we expect tools to support the integration of different engineering artefacts and enable, by default, effective development with quality requirements in mind – such as safety, security, reliability, dependability, sustainability, trustworthiness, and interoperability. New methods and tools will need to be developed to further facilitate software interaction with other elements in a system engineering context (cf. Chapter 2.3 **Architecture and Design: Methods and Tools**).

Artificial intelligence is a technology that holds a great potential in dealing with large amount of data, and potentially could be used for understanding complex systems. In this context,

[11] https://www.intel.com/content/www/us/en/developer/articles/technical/efficient-heterogenous-parallel-programming-openmp.html#gs.85zv3a
[12] https://www.khronos.org/sycl/

DRAFT

artificial intelligence could hold potential to automate some daily engineering tasks, moving boundaries of type and size of tasks that are humanly possible in software engineering.

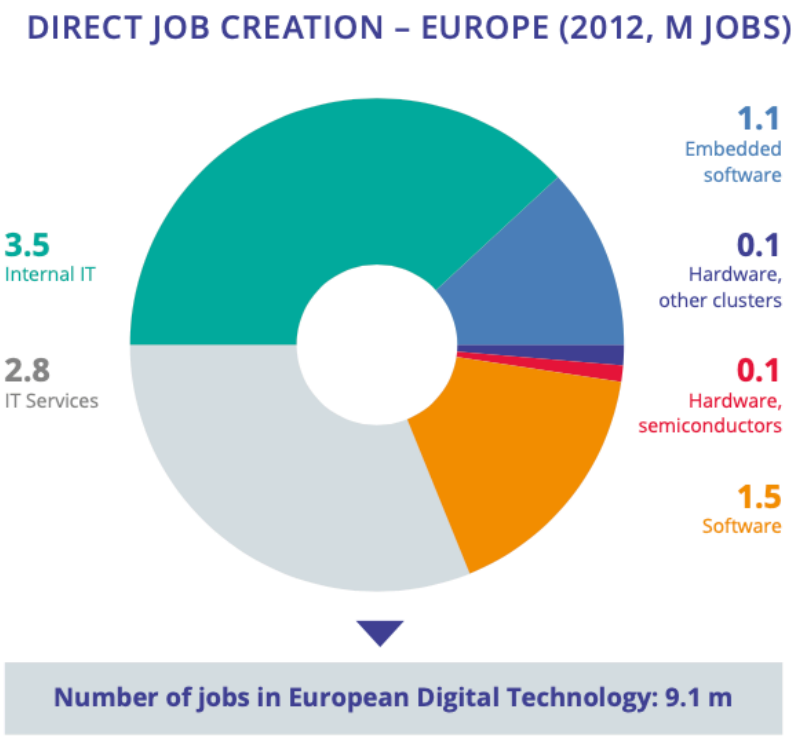

Figure 6 Direct job creation – Europe (2012, m jobs) Source: EU, IDC, Destatis, Roland Berger

(B) Software architectures for optimal edge computing

At the moment, Edge computing lacks proper definition and, including many different types of managed and unmanaged devices, this leads to uncertainty and difficulties on how to efficiently and effectively use software architectures, including aspects as resource, device, and network management (between edge devices as well between edge and fog/cloud), security, useful abstractions, privacy, security, reliability, and scalability. Additionally, automatic reconfiguration, adaptation and re-use face a number of challenges. These challenges are caused by diversity of edge devices and wide range of requirements in terms of Quality of Service (e.g., low latency, high throughput). Finally, sustainability and reliability are difficult to be ensured when trying to prioritize between Quality of Service on the edge and end-to-end system Quality of Service.

Furthermore, the lack of definition also hampers the growing need for energy efficient computing and the development of energy consumption solutions and models across all layers from materials, via software architecture to embedded/application software. Energy efficiency is vital for optimal edge computing.

Lastly, as AI is also moving towards edge (i.e., Edge AI) defining lightweight models and model architectures that can deal with low amount of data available on the edge and still provide good model accuracy are desperately needed. Finally, this limits transfer of common solution patterns, best practices, and reference architectures, as Edge computing scope and configuration requires further clarification and classification.

Since edge devices need to be self-contained, edge software architectures need to support, from one side, virtual machine-like architectures, and from the other side they need to support the entire software lifecycle. The fact that there are many different types of edge

devices would also require an interoperability standard to ensure that they can work together. Innovations in this field should focus on, amongst others, software hardware co-design, virtualisation and container technologies and new standard edge software architecture (middleware).
It is essential to discuss types of quality properties that become more significant as Edge computing is introduced, and based on these, build use cases that profit from quality properties specific to edge computing. There is a need for new approaches that enable early virtual prototyping of edge solutions, as well as approaches that enable verification and validation of quality properties during entire life cycle of edge software systems. One of the possibilities for profiting from Edge is to focus on digital twins to monitor divergences from expected behaviour and implement logic that will benefit from Edge's low latency when making critical decisions, especially in safety critical software systems.

(C) Integration **of** embedded software

To ensure software development is more effective and efficient, it is necessary to place greater focus on integrating embedded software into a fully functional system. First, innovation in continuous system integration must include more effective ways of integrating legacy components into new systems (see also D). Second, for the integration of data and software, the embedded software running in the field has to generate data (such as on run-time performance monitoring, system health, quality of output, compliance to regulations, user interactions) that can be re-used to improve its quality and performance. By improving this, the data and software integration can not only improve the efficiency of embedded software itself, but also the internal coordination and orchestration between components of the system by ensuring a rapid feedback cycle. Third, it is a paramount to enable closer integration of software with the available computing accelerators. This must be done in a way that frees developers from additional effort, while at the same time uses the full potential of heterogeneous computing hardware.

(D) Using abstraction and virtualisation

The recent focus on model-driven software development (or "low code") has sparked a new approach to managing complexity and engineering software. Generating embedded software from higher-level models can improve maintainability and decrease programming errors, while also improving development speed. However, creating and managing models of real systems with an appropriate level of detail that allows for simulation and code generation is a challenge. Managing models and model variability is a necessity if we want to prevent shifting the code legacy problem to a model legacy problem where there are too many models with too much variety.

The core elements of the domain are captured in a language of the domain. The introduction of domain-specific languages (DSLs) and aspect-oriented languages has allowed for the inclusion of aspects and constructs of a target application domain into the languages used to develop embedded software. This abstraction allows for shortening the gap between software engineers and domain experts. We expect innovations in DSLs and tools support to establish a major boost in the efficiency of embedded software development.

The increased level of abstraction allows for more innovation in virtualisation of systems and is a step towards correctness by construction instead of correctness by validation/testing.

DRAFT

Model-based engineering and digital twins of systems are already being used for a variety of goals – such as training, virtual prototyping and log-based fault analysis. Furthermore, they are necessary for supporting transition towards sustainable ECPS. Innovations in virtualisation will allow DSLs to be (semi-)automatically used to generate digital twins with greater precision and more analysis capabilities, which can help us to explore different hardware and software options before a machine is even built, shortening development feedback loops due to such improved tool-supported software development.

(E) Resolving legacy

Legacy software and systems still constitute most of the software running in the world today. It is only natural that the number of legacy increases in future. While it is paramount to develop new and improved techniques for the development and maintenance of embedded software, we cannot ignore the systems currently in operation. New software developed with novel paradigms and new tools will not run in isolation, but rather have to be used increasingly in ecosystems of connected hardware and software, including legacy systems.

There are two main areas for innovation here. First, we need to develop efficient ways of improving interoperability between new and old. With investments of years of development, embedded knowledge and a need to continue operations, we will have to depend on legacy software for the foreseeable future. It is therefore imperative to develop new approaches to facilitating reliable and safe interactions, including wrapping old code in re-usable containers. Second, we must innovate how to (incrementally) migrate, rejuvenate, redevelop and redeploy legacy software, both in isolation and as part of a larger system. We expect innovations in these areas to increase efficiency and effectiveness in working with legacy software in embedded software engineering.

DRAFT

1.3.5.1.3. Key focus areas

The key focus areas in the domain of efficient embedded software engineering include the following:

There is a strong relation between major challenges 1, 2, 3, and 6 below, and chapter 2.3 "Architectures and Design: Methods and Tools“ major challenges 1 and 2 -> link

- Model-based software engineering:
  - Model-based software engineering enabling systems to become part of SoS.
  - Model inference to enable re-use, refactoring and evolution of existing subsystems in SoS.
  - Model-based testing that takes the re-use of uncontrolled systems into account.
  - Embedded software architectures to enable SoS <<link to ch. 1,4>>.

- Digital twinning:
  - Virtualisation as means for dealing with legacy systems.
  - Virtualisation and virtual integration testing (using Digital Twins and specialized design methods, like e.g.., contract based design, for guaranteeing safe and secure updates (cf. Architecture and Design: Methods and Tools Chapter 2.4).

- Approaches to reduce re-release/re-certification time, e.g. model based design, contract based design, modular architectures.
- Distinct core system versus applications and services.
- Design for X (e.g design for test, evolvability and updateability, diagnostics, adaptability).

- Constraint environments:
  - Knowledge-based leadership in design and engineering.
  - Resource planning and scheduling (including multi-criticality, heterogeneous platforms, multicore, software portability).
  - Simulation and Design for software evolution over time, while catering for distinct phases.
  - Exploiting hybrid compute platforms, including efficient software portability.

- Software technology:
  - Virtualisation as tool for efficient engineering.
  - Interface management enabling systems to become part of SoS.
  - Technology for safe and dependable software ecosystems.
  - Artificial intelligence-based tools to support software engineering efforts.
  - co-simulation platforms

- SW engineering tools:
  -
  - integrating embedded AI in software architecture and design.
  - Programming languages for developing large-scale applications for embedded systems.
  - Models & digital twins, also at run-time for maintenance and sustainability.
  - Compilers, code generators, and frameworks for optimal use of heterogeneous computing platforms.
  - co-simulation platforms
  - Tools, middleware and (open) hardware <link to the RISC-V appendix>> with permissible open source licenses …. --> Philippe please

DRAFT

2.3.5.1. Major Challenge 2: continuous integration and delivery

2.3.5.1.1. State of the art

It is fair to assume that most future software applications will be developed to function as a part of a certain platform, and not as standalone components. In some embedded system domains, this idea has been a reality for a decade (e.g., in the AUTomotive Open System Architecture (AUTOSAR) partnership, which was formed in 2003). Increasingly the platforms have to support SoS and IoT integration and orchestration, involving a large amount of diverse small devices. Guaranteeing quality properties of software (e.g., safety and security) is a challenging task, and one that only becomes more complex as the size and distribution of software applications grow, especially if software is not properly designed for its intended

operational context (cf. Chapter 2.3 Architecture and Design: Methods and Tools). Although we are aiming towards continuous integration on the level of IoT and SoS, we are still struggling with the integration of code changes from multiple contributors into a single software system.

One aspect of the problem relates to the design of SoS[13], which are assumed to be composed of independent subsystems but over time have become dependent. Orchestration between the different subsystems, that may involve IoT as well, is an additional issue here. Another aspect relates to the certification of such systems that requires a set of standards. This applies especially for IoT and SoS and it is complicated by the introduction of AI into software systems. Although AI is a software-enabled technology, there are still many issues on the system level when it comes to its integration into software systems. It is particularly challenging to ensure their functional safety and security, and thus to certify such systems. Some of the existing initiatives that are moving towards certification include SAE J3016, which recommends a taxonomy and definitions for terms related to automated driving. Note, that AI may be applied as an engineering tool to simplify certification.

Finally, integration and delivery practices are part of the engineering processes. Although methodologies already exist to achieve this (such as DevSecOps and ChatOps), these mostly relate to software production. With ECPS, continuous integration becomes increasingly more complex, since the products into which the new software modules have to be integrated into are already sold and 'working in the field', often in many different variants (i.e. the whole car fleet of an OEM). Even in domains where the number of variant systems is small, retaining a copy of each system sold at the producing company in order to have an integration target is prohibitive. Thus, virtual integration using model-based design methods (including closed-box models for legacy components) and digital twins used as integration targets as well as for verification & validation by physically accurate simulation are a mandatory asset for any system company to manage the complexity of ECPS and their quality properties. System engineering employing model-based design and digital twins must become a regular new engineering activity.

DRAFT

2.3.5.1.2. Vision and expected outcome

Europe is facing a great challenge with the lack of platforms that are able to adopt embedded applications developed by individual providers into an ecosystem (cf. Reference Architectures and Platforms in Chapter 2.3). The main challenges here are to ensure the adequate functionality of integrated systems (which is partially solved by the microservices approach), while ensuring key quality properties such as performance, safety, and security (see also Major Challenge 6) (which is becoming increasingly complex and neglected as we adopt approaches that facilitate only integration on the functional level). Instrumental for these challenges is the use of integration and orchestration platforms that standardise many of the concerns of the different parts in the SoS, some of which are connected via IoT. In addition, Automated engineering processes will be crucial to ease the integration of parts.
ECPS will become a part of an SoS and eventually SoECPS. SoS challenges like interoperability, composability, evolvability, control, management and engineering demand ECPS to be

[13] R. Kazman, K. Schmid, C. B. Nielsen and J. Klein, "Understanding patterns for system of systems integration," 2013 8th International Conference on System of Systems Engineering, 2013, pp. 141-146, doi: 10.1109/SYSoSE.2013.6575257.

prepared for a life as a part of a SoS (cf. Chapter 1.4 System of systems). Thus precautions at individual ECPS's are necessary to enable cost efficient and trustworthy integration into SoS. Therefore, it is essential to tackle these challenges by good engineering practices: (i) providing sets of recommended code and (system to system) interaction patterns; (ii) avoiding anti-patterns; and (iii) ensuring there is a methodology to support the integration from which the engineers of such systems can benefit. This implies aiming to resolve and pre-empt as many as possible of the integration and orchestration challenges on the platforms design level. It also involves distribution of concerns to the sub systems in the SoS or IoT. Followed by automated engineering processes applying the patterns and dealing with the concerns in standardised ways. Besides this, it is necessary to facilitate communication between different stakeholders to emphasise the need for quality properties of ECPS, and to enable (automated) mechanisms that raise concerns sufficiently early to be prevented while minimising potential losses.

On the development level, it is key to enhance the existing software systems development methodologies to support automatic engineering, also to automate the V&V processes for new features as they are being introduced into the system. This might need the use of AI in the V&V process. At this level, it is also necessary to use of software system architecture in the automation of V&V and other engineering practices, to manage the complexity that arises from such integration efforts (also see Major Challenge 3 below).

DRAFT

2.3.5.1.3. Key focus areas

The key focus areas identified for this challenge include the following:

- Continuous integration of embedded software:
  - Model based design and digital twins to support system integration (HW/SW) and HW/SW co-development (increasingly new technologies have to be integrated).
  - Applying automation of engineering, taking architecture, platforms and models into account.
  - Virtualisation and simulation as tools for managing efficient integration and validation of configurations, especially for shared resources and other dependability issues.
  - Application of integration and orchestration practices to ensure standard solutions to common integration problems
  - Integration and orchestration platforms and separation of concerns in SoS and IoT.

- Verification and validation of embedded software:
  - (Model) test automation to ensure efficient and continuous integration of CPSs.
  - Enabling secure and safe updates (cf. Major Challenge 3) and extending useful life (DevOps).
  - Continuous integration, verification and validation (with and without AI) enabling continuous certification with automated verification & validation

(especially the focus on dependability), using model-based design technologies and digital twins; also when SoS and IoT are involved.
- Certification of safety-critical software in CPSs.

3.3.5.1. Major Challenge 3: lifecycle management

3.3.5.1.1. State of the art

Complex systems such as airplanes, cars and medical equipment are expected to have a long lifetime, often up to 30 years. The cost of keeping these embedded systems up to date, making them relevant for the everyday challenges of their environment is often time-consuming and costly. This is becoming more complex due to most of these systems becoming cyber-physical systems, meaning that they link the physical world with the digital world, and are often interconnected with each other or to the internet. With more and more functionalities being realized by embedded software, over-the-air updates – i.e. deploying new, improved versions of software-modules unto systems in the field – become an increasingly relevant topic. Apart from updates needed for error and fault corrections, performance increases and even the implementation of additional functionalities – both optional or variant functionalities that can be sold as part of end-user adaptation as well as completely new functionalities that are needed to respond to newly emerging environmental constraints (e.g. new regulations, new features of cooperating systems). Such update capabilities perfectly fit and even are required for the 'continuous development and integration' paradigm.

Embedded software also has to be maintained and adapted over time, to fit new product variants or even new product generations and enable updateability of legacy systems. If this is not effectively achieved, the software becomes overly complex, with prohibitively expensive maintenance and evolution, until systems powered by such software are no longer sustainable. We must break this vicious cycle and find new ways to create software that is long-lasting and which can be cost-efficiently evolved and migrated to use new technologies. Practical challenges that require significant research in software sustainability include: (i) organisations losing control over software; (ii) difficulty in coping with modern software's continuous and unpredictable changes; (iii) dependency of software sustainability on factors that are not purely technical; (iv) enabling "write code once and run it anywhere" paradigm.

3.3.5.1.2. Vision and expected outcome

As software complexity increases, it becomes more difficult for organisations to understand which parts of their software are worth maintaining and which need to be redeveloped from scratch. Therefore, we need methods to reduce the complexity of the software that is worth maintaining, and extracting domain knowledge from existing systems as part of the redevelopment effort. This also relates to our inability to monitor and predict when software quality is degrading, and to accurately estimate the costs of repairing it. Consequently, sustainability of the software is often an afterthought. This needs to be flipped around – i.e. we need to design "future-proof" software that can be changed efficiently and effectively, or at least platforms for running software need to either enable this or force such way of thinking.

DRAFT

As (embedded) software systems evolve towards distributed computing, SoS and microservice-based architectural paradigms, it becomes even more important to tackle the challenges of integration at the higher abstraction levels and in a systematic way. Especially when SoS or IoT is involved, it is important to be able to separate the concerns over the subsystems.
The ability of updating systems in the field in a way that safety of the updated systems as well as security of the deployment process is maintained will be instrumental for market success of future ECPS. Edge-to-cloud continuum represents an opportunity to create software engineering approaches and engineer platforms that together enable deployment and execution of the same code anywhere on this computing continuum.
The ability of keeping track of system parameters like interface contracts and composability requires a framework to manage these parameters over the lifetime. This will enable the owner of the system to identify at any time how the system is composed and with what functionality.
Instead of focusing just on the efficiency of embedded software engineering, we already see that the field is evolving into direction of cyber physical systems (cf. Chapter 2.3 **Architecture and Design: Methods and Tools**), and software is one element of engineering.

Many software maintenance problems are not actually technical but people problems. There are several socio-technical aspects that can help, or hinder, software change. We need to be able to organise the development teams (e.g. groups, open source communities) in such a way that it embraces change and facilitates maintenance and evolution, not only immediately after the deployment of the software but for any moment in software lifecycle, for the decades that follow, to ensure continuity. We need platforms that are able to run code created for different deployment infrastructure, without manual configuration.

The expected outcome is that we are able to keep embedded systems relevant and sustainable across their complete lifecycle, and to maintain, update and upgrade embedded systems in a safe and secure, yet cost-effective way.

DRAFT

3.3.5.1.3. Key focus areas

The key focus areas identified for this challenge include the following.

- Rejuvenation of systems:
  - Software legacy and software rejuvenation to remove technical debt (e.g. software understanding and conformance checking, automatic redesign and transformation).
  - Continuous platform-agnostic integration, deployment and migration.
  - End-of-life and evolving off-the-shelve/open source (hardware/software).

- Managing complexity over time:
  - Interplay between legacy software and new development approaches.
  - Vulnerability of connected systems.
  - Continuous certification of updates in the field (reduce throughput time).
  - Intelligent Diagnostics of systems in the field (e.g. guided root cause analysis)

- Managing configurations over time:
  - Enable tracking system configurations over time.
  - Create a framework to manage properties like composability and system orchestration.

- Evolvability of embedded software:
  - Technology, including automation of engineering and the application of integration and orchestration platforms, for keeping systems maintainable, adaptable and sustainable considering embedded constraints with respect to resources, timing and cost.
  - Embedded software architectures to enable SoS.

4.3.5.1. Major Challenge 4: embedding Data Analytics and Artificial Intelligence

4.3.5.1.1. State of the art

For various reasons – including privacy, energy efficiency, latency and embedded intelligence – processing is moving towards edge computing, and the software stacks of embedded systems need to support more and more analysis of data captured by the local sensors and to perform AI-related tasks. As detailed in the Chapter Artificial Intelligence, Edge computing and Intelligence on the Edge, non-functional constraints of embedded systems, such as timing, energy consumption, low memory and computing footprint, being tamperproof, etc., need to be taken into account compared to software with similar functionalities when migrating these from cloud to edge. Furthermore, Quality, Reliability, Safety and Security Chapter states that key quality properties when embedding of AI components in digitalized ubiquitous systems are determinism, understanding of nominal and degraded behaviours of the system, their certification and qualification, and clear liability and responsibility chains in the case of accidents. When engineering software that contains AI-based solutions, it is important to understand the challenges that such solutions introduce. AI contributes to challenges of embedded software, but itself it does not define them exclusively, as quality properties of embedded software depend on integration of AI-based components with other software components.
For efficiency reasons, very intensive computing tasks (such as those based on deep neural networks, DNNs) are being carried out by various accelerators embedded in systems on a chip (SoCs). Although the "learning" phase of a DNN is still mainly done on big servers using graphics processing units (GPUs), local adaptation is moving to edge devices. Alternative approaches, such as federated learning, allow for several edge devices to collaborate in a more global learning task. Therefore, the need for computing and storage is ever-increasing, and is reliant on efficient software support.

The "inference" phase (i.e. the use after learning) is also requiring more and more resources because neural networks are growing in complexity exponentially. Once carried out in embedded GPUs, this phase is now increasingly performed on dedicated accelerators. Most middle and high-end smartphones have SoC embedding one of several AI accelerators – for example, the Nvidia Jetson Xavier NX is composed of six Arm central processing units (CPUs), two inference accelerators, 48 tensor cores and 384 Cuda cores. Obtaining the best of the

DRAFT

heterogeneous hardware is a challenge for the software, and the developers should not have to be concerned about where the various parts of their application are running.

Once developed (on servers), a neural network has to be tuned for its embedded target by pruning the network topology using less precision for operations (from floating point down to 1-bit coding) while preserving accuracy. This was not a concern for the "big" AI development environment providers (e.g., Tensorflow, PyTorch, Caffe2, Cognitive Toolkit) until recently. This has led to the development of environments designed to optimise neural networks for embedded architectures[15]: for the move towards the Edge.

Most of the time the learning is done on the cloud. For some applications/domains, making a live update of the DNN characteristics is a sought-after feature, including all the risks of security, interception. Imagine the consequences of tampering with the DNN used for a self-driving car! A side-effect of DNN is that intellectual property is not in a code or algorithm, but rather lies in the network topology and its weights, and therefore needs to be protected.

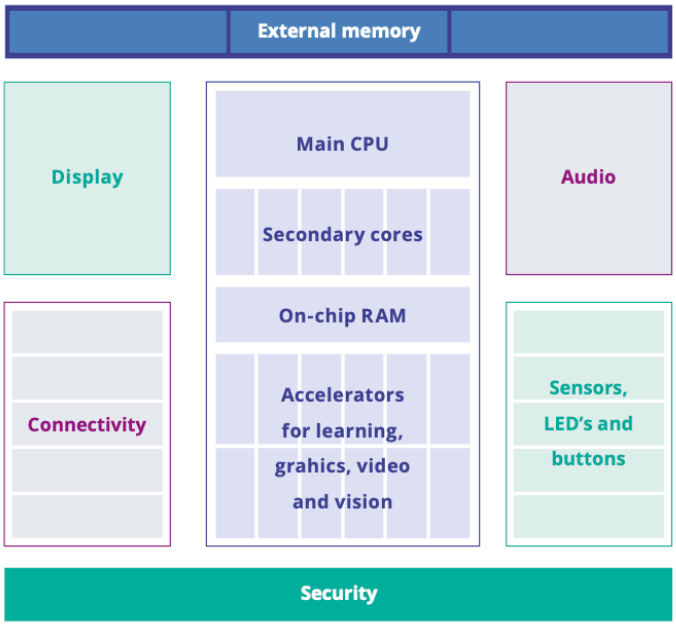

Figure 7 Data analytics and Artificial Intelligence require dedicated embedded hardware architectures

4.3.5.1.2. Vision and expected outcome

European semiconductor providers lead a consolidated market of microcontroller and low-end microprocessor for embedded systems, but are increasing the performance of their hardware, mainly driven by the automotive market and the increasing demand for more performing AI for advanced driver-assistance systems (ADAS) and self- driving vehicles. They are also moving towards greater heterogeneity by adding specialised accelerators. On top of this, Quality, Reliability, Safety and Security Chapter lists personalization of mass products and resilience to cyber-attacks, as the key advantage and the challenge characterizing future products. Embedded software needs to consider these and find methods and tools to manage

[15] Such as N2D2, https://github.com/CEA-LIST/ N2D2

their effects on quality properties of software that integrates them. Also, embedded software engineering will need to ensure interoperability between AI-based solutions and non-AI parts.

In this context, there is a need to provide a programming environment and libraries for the software developers. A good example here is the interchange format ONNX, an encryption format for protection against tampering or reverse engineering that could become the foundation of a European standard. Beside this, we also need efficient libraries for signal/image processing for feeding data and learning into the neural network, abstracting from the different hardware architectures. These solutions are required to be integrated and embedded in ECPS, along with significant effort into research and innovation in embedded software.

4.3.5.1.3. Key focus areas

The key focus areas identified for this challenge include the following:

- Federated and distributed learning:
  - Create federated learning at the edge in heterogeneous distributed systems (analysis, modelling and information gathering based on local available information).
  - Federated intelligence at the edge (provide context information and dependability based on federated knowledge).

- Embedded Intelligence:
  - Create a software AI framework to enable reflecting and acting on the systems own state.
  - Dynamic adaption of systems when environment parameters and sensors like IoT devices are changing.

- Data streaming in constraint environments:
  - Feed streaming data into low-latency analysis and knowledge generation (using context data to generate relevant context information).

- Embedding AI accelerators:
  - Accelerators and hardware/software co-design to speed up analysis and learning (e.g. patter analysis, detection of moves (2D and 3D) and trends, lighting conditions, shadows).
  - Actual usage-based learning applied to accelerators and hardware/software co-design (automatic adaptation of parameters, adaptation of dispatch strategies, or use for new accelerators for future system upgrades).

5.3.5.1. Major Challenge 5: support for sustainability by embedded software

5.3.5.1.1. State of the art

DRAFT

The complete power demand in the whole ICT market currently accounts from 5% to 9% of the global power consumption[17]. The ICT electricity demand is rapidly increasing and it could go up to nearly 20% in 2030**Error! Bookmark not defined.**. Compared to estimated power consumption of future large data centres, embedded devices may seem to be a minor problem. However, when the devices powered by batteries still have a significant environmental impact. Energy efficient embedded devices produce less hazardous waste and last longer time without need to be replaced.

The growing demand for ultra-low power electronic systems has motivated research into device technology and hardware design techniques. Experimental studies have proven that the hardware innovations for power reduction can be fully exploited only with proper design of the upper layer software. The same applies to software power and energy modelling and analysis: the first step towards the energy reduction is complex due to the inter- and intra-dependencies of processors, operating systems, application software, programming languages and compilers. Software design and implementation should be viewed from a system energy conservation angle rather than as an isolated process.
For sustainability, it is critical to understand quality properties of software. These include in the first place power consumption, and then other related properties (performance, safety, security, and engineering-related effort) that we can observe in the context of outdated or inadequate software solutions and indicators of defected hardware. Power reduction strategies are mainly focusing on processing, storage, communication, and sometimes on other (less intelligent) equipment.
For the future embedded software developers, it is crucial to keep in touch with software development methodologies focused to sustainability, such as green computing movement or sustainable programming techniques. In the domain of embedded software, examples include the remaining useful life of the device estimation, the network traffic and latency time optimization, the process scheduling optimization or energy efficient workload distribution.

DRAFT

5.3.5.1.2. Vision and expected outcome

The concept of sustainability is based on three main principles: the ecological, the economical and the social. The ideal environmentally sustainable (or green) software in general requires as little hardware as possible, it is efficient in the power consumption, and its usage leads to minimal waste production. An embedded software designed to be adaptable for future requirements without need to be replaced by a completely new product is an example of environmentally, economically, and socially sustainable software.
To reach the sustainability goal, the embedded software design shall focus also on energy-efficient design methodologies and tools, energy efficient and sustainable techniques for embedded software and systems production and to development of energy aware applications and frameworks for the IoT, wearable computing or smart solutions or other application domains.
It is evident that energy/power management has to be analysed with reference to the context, underlying hardware and overall system functionalities. The coordinated and concentrated efforts of a system architect, hardware architect and software architect should help introduce energy-efficient systems (cf. Chapter 2.3, Architecture and Design: Methods

[17] https://www.enerdata.net/publications/executive-briefing/between-10-and-20-electricity-consumption-ict-sector-2030.html

and Tools). The tight interplay between energy-oriented hardware, energy-aware and resource-aware software calls for innovative structural, functional and mathematical models for analysis, design and run-time. Model-based software engineering practices, supported by appropriate tools, will definitely accelerate the development of modern complex systems operating under severe energy constraints. It is crucial to notice the relationship between power management and other quality properties of software systems (e.g., under certain circumstances it is adequate to reduce the functionality of software systems by disabling certain features, which results in significant power savings). From a complementary perspective, when software is aware of the available hardware and energy resources, it enables power consumption optimisation and energy saving, being able to configure the hardware resources, to activate/deactivate specific hardware components, increase/decrease the CPU frequency according to the processing requirements, partition, schedule and distribute tasks.

Therefore, in order to enable and support sustainability through software, software solutions need to be reconfigurable in the means of their quality. There have to exist strategies for HW/SW co-design and accelerators to enable such configurations. For this to be possible, software systems need to be accompanied with models of their quality properties and their behaviour, including the relationship between power consumption and other high level quality properties. This will enable balancing mechanisms between local and remote computations to reduce communication and processing energy.

Models (digital twins) should be aware of energy use, energy sources and the sustainability of the different sources. An example of this in SoS are solar cells that give different amounts of energy dependent on daytime and weather conditions.

DRAFT

5.3.5.1.3. Key focus areas

The following key focus areas have been identified for this challenge:

- Resource-aware software engineering.
- Tools and techniques enabling the energy-efficient and sustainable embedded software design.
- Development of energy-aware and sustainable frameworks and libraries for the main embedded software application areas (e.g. IoT, Smart Industry, Wearables).
- Management of computation power on embedded hardware.
    - Management of energy awareness of embedded hardware, embedded software with respect to, amongst others, embedded high-performance computing (HPC).
- Composable efficient abstractions that drive sustainable solutions while optimising performance
    - Enabling technologies for the second life of (legacy) cyber-physical systems.
    - Establish relationships between power consumption and other quality properties of software systems, including engineering effort (especially in cases of computing-demanding simulations).
    - Digital twins can contribute in management quality properties of software systems with goal of reducing power consumption, as the major contributing factor to green deal, enabling sustainability.

6.3.5.1. Major Challenge 6: software reliability and trust

6.3.5.1.1. State of the art

Two emerging challenges for reliability and trust in ECPS relate to computing architectures and the dynamic environment in which ECPS exist. The first challenge is closely related to the end of Dennard scaling[18]. In the current computing era, concurrent execution of software tasks is the main driving force behind the performance of processors, leading to rise of multicore and manycore computing architectures. As the number of transistors on a chip continues to increase (Moore's law is still alive), industry has turned to a heavier coupling of software with adequate computing hardware, leading to heterogeneous architectures. The reasons for this coupling are the effects of dark silicon[19] and better performance-to-power ratio of heterogeneous hardware with computing units specialised for specific tasks achieves. The main challenges for using concurrent computing systems in embedded systems remain: (i) hard-to-predict, worst-case execution time; and (ii) testing of concurrent software against concurrency bugs[20].

The second challenge relates to the dynamic environment in which ECPS execute. On the level of systems and SoS, architectural trends point towards platform-based designs – i.e. applications that are built on top of existing (integration and/or middleware) platforms. Providing a standardised "programming interface" but supporting a number of constituent subsystems that is not necessarily known at design time, and embedding reliability and trust into such designs, is a challenge that can be solved only for very specialised cases. The fact that such platforms – at least on a SoS level – are often distributed further increases this challenge.

On the level of systems composed from embedded devices, the most important topics are the safety, security, and privacy of sensitive data. Security challenges involve: (i) security of communication protocols between embedded nodes, and the security aspects on the lower abstraction layers; (ii) security vulnerabilities introduced by a compiler[21] or reliance on third-party software modules; and (iii) hardware-related security issues[22]. It is necessary to observe security, privacy and reliability as quality properties of systems, and to resolve these issues on a higher abstraction level by design[23], supported by appropriate engineering processes including verification. << link to ch. 2.4>> <<link to ch. 2.3>>

[18] John L. Hennessy and David A. Patterson. 2019. A new golden age for computer architecture. Commun. ACM 62, 2 (February 2019), 48–60. DOI:https://doi.org/10.1145/3282307

[19] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark silicon and the end of multicore scaling. In Proceedings of the 38th annual international symposium on Computer architecture (ISCA '11). Association for Computing Machinery, New York, NY, USA, 365–376. DOI:https://doi.org/10.1145/2000064.2000108

[20] F. A. Bianchi, A. Margara and M. Pezzè, "A Survey of Recent Trends in Testing Concurrent Software Systems," in IEEE Transactions on Software Engineering, vol. 44, no. 8, pp. 747-783, 1 Aug. 2018, doi: 10.1109/TSE.2017.2707089.

[21] V. D'Silva, M. Payer and D. Song, "The Correctness-Security Gap in Compiler Optimization," 2015 IEEE Security and Privacy Workshops, 2015, pp. 73-87, doi: 10.1109/SPW.2015.33.

[22] Moritz Lipp, Vedad Hadžić, Michael Schwarz, Arthur Perais, Clémentine Maurice, and Daniel Gruss. 2020. Take A Way: Exploring the Security Implications of AMD's Cache Way Predictors. In Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS '20). Association for Computing Machinery, New York, NY, USA, 813–825. DOI:https://doi.org/10.1145/3320269.3384746

[23] Dalia Sobhy, Leandro Minku, Rami Bahsoon, Tao Chen, Rick Kazman, Run-time evaluation of architectures: A case study of diversification in IoT, Journal of Systems and Software, Volume 159, 2020, 110428, ISSN 0164-1212, https://doi.org/10.1016/j.jss.2019.110428.

DRAFT

6.3.5.1.2. Vision and expected outcome

European industry today relies on developed frameworks that facilitate production of highly complex embedded systems (for example, AUTOSAR in the automotive industry). But beyond that, platforms with these qualities are exactly what is needed for powering European digital infrastructures.

The ambition here is to reach a point where such software system platforms are mature and available to a wider audience. These platforms need to enable faster harvesting of hardware computing architectures that already exist and provide abstractions enabling innovators and start-ups to build new products quickly on top of them. For established businesses, these platforms need to enable shorter development cycles while ensuring their reliability and providing means for verification & validation of complex systems. The purpose of building on top of these platforms is ensuring, by default, a certain degree of trust for resulting products. This especially relates to new concurrent computing platforms, which hold promise of great performance with optimised power consumption.

Besides frameworks and platforms that enable easy and quick development of future products, the key enabler of embedded software systems is their interoperability and openess. In this regard, the goal is to develop and make available to a wider audience, software libraries, software frameworks and reference architectures that enable interoperability and integration of products developed on distributed computing architectures. They also need to ensure, by design, the potential for monitoring, verification, testing and auto-recovery of embedded systems. One of the emerging trends to help achieving this is the use of digital twins. Digital twins are particularly suitable for the verification of safety-critical software systems that operate in dynamic environments. However, development of digital twins remains an expensive and complex process, which has to be improved and integrated as part of the standard engineering processes <<link to ch 2.3 / Challenge 2>>

We envision an open marketplace for software frameworks, middleware, and digital twins that represents a backbone for the future development of products. While such artefacts need to exploit the existing software stacks and hardware, they also need to support correct and high-quality software by design. Special attention is required for Digital Twin simulations of IoT devices to ensure reliability and trust in operating in real life.

6.3.5.1.3. Key focus areas

Focus areas of this challenge are related to quality aspects of software. For targets such as new computing architectures and platforms, it is crucial to provide methodologies for development and testing, as well as for the team development of such software. These methodologies need to take into account the properties, potentials and limitations of such target systems, and support developers in designing, analysing and testing their implementations. As it is fair to expect that not all parts of software will be available for testing at the same time, it is necessary to replace some of the concurrently executing models using simulation technologies. Finally, these achievements need to be provided as commonly available software modules that facilitate the development and testing of concurrent software.

DRAFT

The next focus area is testing of systems against unexpected uses, which mainly occurs in systems with a dynamic execution environment. It is important here to focus on testing of self-adapting systems where one of the predominant tools is the simulation approach, and more recently the use of digital twins.

However, all these techniques are not very helpful if the systems are not secure and reliable by design. Therefore, it is necessary to investigate platforms towards reliability, security and privacy, with the following challenges

- Reliable software on new hardware including edge, fog and cloud processing: (co) verification of distributed, also heterogenous systems
- Verification and validation of ML models

- Robustness against unexpected uses:
  - Trustworthy, secure, safe, privacy-aware.
  - Validating self-adapting systems for example through simulation.

- Security and privacy as a service:
  - To become part of the software architecture.
  - Means and techniques for continuous system monitoring and self-monitoring.

DRAFT

1.3.6. TIMELINE

The following table illustrates the roadmaps for Embedded Software and Beyond. The assumption is that topic in the cell means that technology should be ready (TRL 9–10) in that timeframe.

MAJOR CHALLENGE	TOPIC	SHORT TERM (2022-2026)	MEDIUM TERM (2027-2030)	LONG TERM (2031-2036)
Major Challenge 1: efficient engineering of embedded software	Topic 1.1: Modelling-based software engineering	Model-based software engineering enabling systems to become part of SoS	Model inference to enable re-use of existing subsystems in SoS	Model-based testing taking re-use of uncontrolled SoS into account
	Topic 1.2 Digital twinning	?	?	?
	Topic 1.3: Constraint environments	• Resource planning and scheduling • Design for	Embedded software architectures to enable SoS	Exploiting hybrid computer platforms, including efficient software portability

		software evolution over time		
	Topic 1.4: Software technology	• Virtualisation as tool for efficient engineering • Technology for safe and dependable software ecosystems	Interface management enabling systems to become part of SoS	• Develop new software architectures for edge computing • Artificial intelligence to assist and support efforts in software engineering
	Topic 1.5: Software engineering tools	co-simulation platforms	Middleware controlling dynamically embedded (mobile) hardware solutionsCompilers and link to new hardware	Programming languages for developing large-scale applications for embedded SoS
Major Challenge 2: continuous integration of embedded software	Topic 2.1: Continuous integration	DevOps modelling Virtualisation	Simulation on a virtual platform	Digital twin Model-based engineering based on digital twins
	Topic 2.2: Verification and validation	Virtualisation of test platform	Model-based testing	Integration & orchestration platforms for IoT and SoS
Major Challenge 3: lifecycle management of embedded software	Topic 3.1: Rejuvenation of existing systems	• Software legacy and software rejuvenation • Design for rejuvenating systems in a later phase	End-of-life and evolving off-the-shelve/open source solutions	• The cloud-for-edge continuum - "Write once, run anywhere" on this computing continuum • Composability, properties contracts and orchestration systems • Interoperability: must be ensured in integration platforms
				•
	Topic 3.2: Managing complexity over time	Diagnostics of systems in the field	Continuous certification	Interplay between legacy

DRAFT

	Topic 3.3: Managing Configurations over time	?	?	?
	Topic 3.4: Evolvability of embedded software	Adaptable embedded software	Dynamical embedded software	• Autonomous embedded software Autonomous processes (IoT & edge embedded HW/SW co-design)
Major Challenge 4: embedding data analytics and AI	Topic 4.1: Federated learning	Create federated learning at the edge in heterogeneous distributed systems	Federated intelligence at the edge	• Safe, trustworthy & explainable AI • AI is playing several key roles in innovation, e.g. as a tool for SW development/engineering Embedded intelligence
	Topic 4.2 Embedded Intelligence	?	?	?
	Topic 4.3: Data streaming in constraint environments	Feed streaming data into low-latency analysis and knowledge generation	?	?
	Topic 4.4: Embedding AI accelerators	Accelerators and hardware/software co-design to speed up analysis and learning	Actual usage based learning applied for accelerators and hardware/software co-design	Use of AI in autonomous systems
Major Challenge 5: support for sustainability by embedded software	Topic 5.1: Green-aware systems	?	Design for green-aware products	?
	Topic 5.2: resource-aware software engineering	Integration of green-aware in software integration		
	Topic 5.3: Tools for energy efficient SW design	Rejuvenation technologies	Design for extending lifetime	Digital twins that support green deal and enable sustainability(e.g. contain power models)
	Topic 5.4 :Energery aware frameworks & libraries	?	?	?

DRAFT

	Mgmnt copmutation power on embedded HW	?	?	?
	Composable efficient abstraction	?	?	?
Major Challenge 6: embedding reliability and trust	Topic 6.1: Reliability of software and new hardware	• Code coverage of reliability tooling and porting • Simulation and mock-up based approaches for handling concurrency	Embed reliability on software architecture level	• Use of quantum computing • IoT digital twin simulation • Validation and verification through simulation and mock-up based approaches for handling concurrency
	Topic 6.2: Robustness (trustworthy, secure, safe, privacy-aware)	• Trustworthy, secure, safe, privacy-aware • Testing self-adapting systems using simulation	Define a maturity model for robustness of embedded software and beyond	
	Topic 6.3: Security and privacy as a service	Design for security and privacy as a service	Architecture for security and privacy as a service	

DRAFT

1.3.7. SYNERGY WITH OTHER THEMES

Opportunities for joint research projects, including groups outside and within the ECS community, can be expected in several sections of the Application Chapters, the Chapters in the technology value stack and with cross sectional Chapters. There are strong interactions with the **System of Systems** Chapter. In the System of Systems Chapter, a reasoning model for system architecture and design is one of the main challenges. Part of system architecture and design is the division in which the system functions will be solved in hardware, and which will be solved in Embedded Software and Beyond. Embedded Software can be divided into two parts: software enabling the hardware to perform, and software implementing certain functionalities. Furthermore, there are connections with the cross-technology Chapters, **Artificial Intelligence, Edge Computing and Intelligence on the Edge, Architecture and**

Design: Methods and Tools, and Quality, reliability, safety and cybersecurity. With respect to AI, using AI as a technology and software components powered by AI in embedded solutions will be part of the **Embedded Software and Beyond** Chapter, while innovating AI will be part of the **AI, Edge Computing and Intelligence on the Edge** Chapter, and discussing its quality properties will be part of the **Quality, reliability, safety and cybersecurity**. With respect to the **Architecture and Design: Methods and Tools Chapter, all methods and tools** belong there, while **Embedded Software and Beyond** focuses on development and integration methodologies. The challenges of preparing useful embedded solutions will be part of the **System of Systems** Chapter and the **Embedded Software and Beyond** Chapter. The embedded software solutions for new computing devices, such as quantum computing, will be part of the **Long-Term Vision** Chapter.

Related to the Health and Wellbeing chapter, the digital transformation in the healthcare industry (see Section 3.4.1) causes software to play an increasingly critical role. More systems and devices are connected to the cloud to collect and combine data and provide SaaS solutions. At the same time, update cycles become shorter, while product and platform lifetimes become longer. These trends lead to the following implications on (embedded) software:

- *Cybersecurity is a crucial element in (embedded) software design.*
- *The distributed nature of software across the system-edge-cloud continuum needs non-conventional software design and test strategies.*
- *Wearable devices often require energy-efficient software design*
- *Shorter release cycles require a higher level of efficiency and automation in the software engineering process, while maintaining the quality standard that is expected in safety-critical systems.*
- *Legacy code becomes a growing challenge in larger health equipment.*

DRAFT