# NESSI

Networked European Software and Services Initiative

# SOFTWARE CONTINUUM

Recommendations for

ICT Work Programme 2018+

*11 May 2016 (Release 2)*

# 1. Introduction

A major transformation is currently under way towards the **Digital Information Society and Economy 2.0**[1]. More and more products, services, processes, and things are being digitized, virtualized and connected. Software will be the essential driver and enabler of innovation and value across these connected entities, spanning from sensors, actuators and processors within everyday objects (Internet of Things, Cyber Physical Systems), over virtualized hardware such as networks, storage and computing resources (Cloud and Fog), to fully digitized business processes and products (Industry 4.0).

Software will be defining everything, from our communication networks to the horsepower of our cars. Leveraging the opportunity of this new Digital Information Society and Economy, thus, will significantly depend on our fundamental ability to create, manage and evolve the software that controls the connected entities in the Digital Information Society and Economy 2.0. As software is spanning a broad range of technology domains and application domains, we require **substantially new and generic software engineering principles, methods and tools** that are applicable across those broad domains. Thereby principle solutions should be developed once and reused across domains. Building software that rests on generic, shared, common principles will help contributing to compatibility of the solutions across domains.

The NESSI view expressed in this document states requirements towards such new and generic software engineering principles, methods and tools. NESSI anticipates two main streams for software creation, management and evolution that will ultimately converge towards a **Software Continuum**. The Software Continuum brings together **self-managing software**, which is able to carry out tasks without human intervention, and **citizen software**, which represents advances in user empowerment concerning software "programming". From a business perspective, the Software Continuum offers a significantly extended range for **value creation**, ranging from software commercially developed by professional developers, via simple software systems developed by novices, up to citizen "programmed" software servicing the specific needs of individuals.

Digital transformation will be a major game changer. Across all sectors – whether in the automotive industry, the chemical industry or the financial industry – software is becoming an essential building block: "**Software Is Eating the World!**" New players are entering the market and attack established companies with highly scalable platform-based business models. Today, the world's largest bookseller is a software company; the world's largest taxi company does not own a single car and the world's largest provider for rooms and apartments has no single hotel of its own. In short: sooner or later, Europe's industrial wealth, which is traditionally strongly based on tangible products and services, will be at stake too.

The **Digital Single Market (DSM)** is therefore of strategic importance. It provides the political and strategic means for ensuring efficient market access and rapid scalability for new smart services. The Software Continuum is the key step to advance and deploy software engineering principles, techniques and tools across all technology and application domains, thus making the DSM a reality. The DSM together with the Software Continuum will thus create numerous opportunities for start-ups and small and medium-sized enterprises, as well as end-users and prosumers. It will strengthen Europe in seizing the opportunities of the Digital Information Society and Economy 2.0.

This document provides the NESSI recommendations towards the ICT Work Programme 2018 – 2019, focussing on the key role that generic software engineering principles, techniques and

[1] See NESSI Prospectus, June 2013 (http://www.nessi-europe.com/Files/Private/NESSI_Prospectus_2013.pdf)

tools will play. It argues for key software engineering challenges to be addressed by research and innovation actions towards realizing such Software Continuum. To avoid replicating investments and research effort, **NESSI advocates setting up centralized activities on generic software engineering principles, techniques and tools, which may feed complementary more domain specific activities.**

## 2. Software Continuum

The Software Continuum is the logical result of the evolution of two main trends of how future software-intensive systems are engineered and evolved: (1) self-managing systems, (2) citizen-programmed systems. These two trends will exploit software and hardware technologies that have reached significant maturity as key enabler for further innovations. Such innovations will allow combining advanced self-managing software transparently with user centric and controlled citizen software with the user highly in the loop. Such technologies include, among others, cognitive software, emotion recognition, intentional programming, deep learning, and open hardware and software platforms.

Figure 1 shows how the maturation of these technologies will push towards a Software Continuum. It shows today's technologies and indicates when emerging technologies will have reached significant maturity to be exploited.

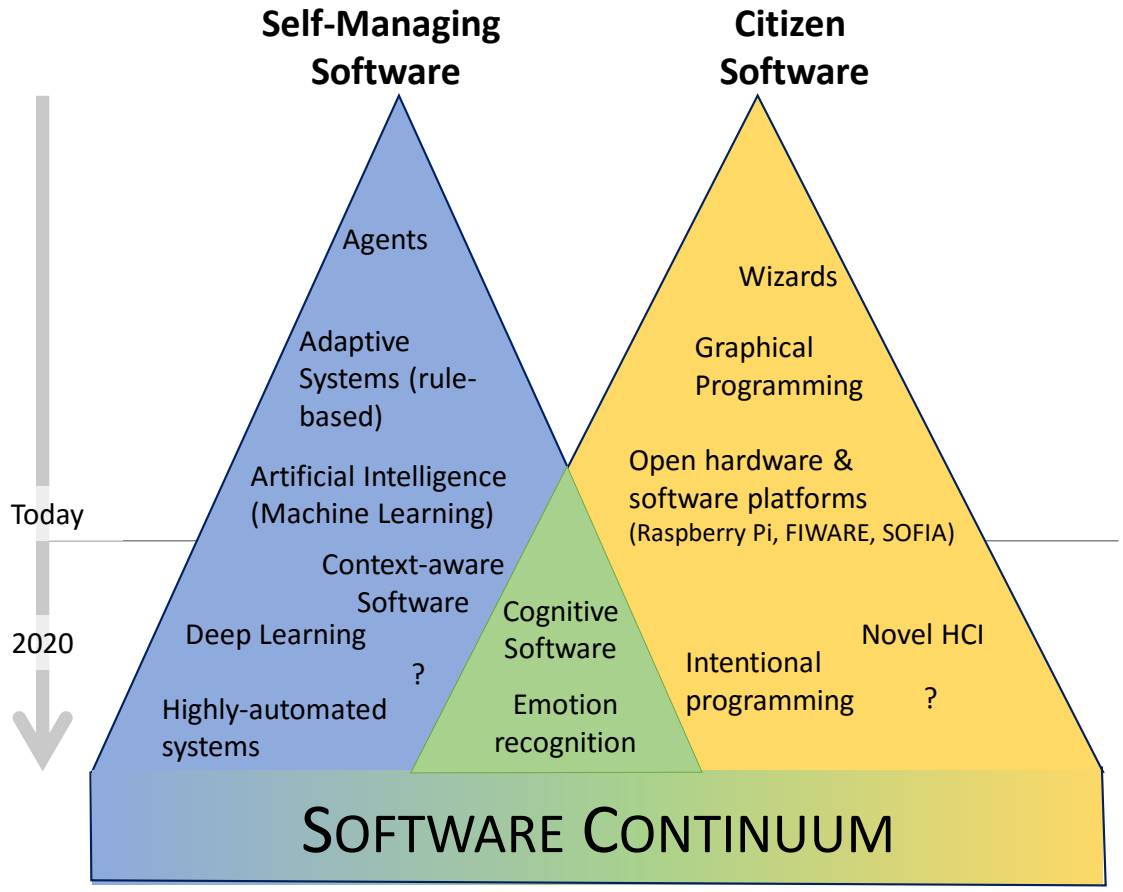


Figure 1. The Software Continuum (dates indicate year when technology is expected to become mature)

The Software Continuum envisions two converging trends, which will mutually benefit and influence each other.

**Self-Managing software:** The software-intensive entities of the Software Continuum will co-ordinate their actions to carry out basic tasks without human intervention. This will involve a high level of automation to deliver self-managing software which will facilitate the adaptation of entities to dynamically changing contexts (such as other elements to coordinate with), and which will cope with unplanned and often unforeseen situations, as well as hardware and software failures. This level of automation will represent a major step forward from existing adaptive software systems (which mainly use pre-defined adaptation strategies, the "known unknowns") and will include (deep) learning, cognitive software and emotion-awareness, ultimately leading to fully automated software (able to handle the "unknown unknowns").

**Citizen software:** Open platforms will emerge that support the citizen-driven programming of software-intensive entities in the Software Continuum. They will include capabilities such as customizing and orchestrating software services, but will go beyond the current techniques and tools which work in the technical solution space (such as concrete software components and code), instead in the problem space (such as requirements or system goals). Users will be able to describe their needs in problem space terms, as rather than the current machine recognizable, solution space terms. This will ultimately lead to the notion of citizen software, where each citizen will be empowered to produce software.

Self-managing software will be a key enabler of citizen software, as it takes away the burden of software optimization, configuration, quality assurance, repair, evolution and adaptation from the person developing the software. As an example, citizen software may automatically deploy itself on the most suitable computing platform. In turn, as citizen-driven programming will happen on the level of the problem space. This will provide self-managing software with better insights into user requirements and their continuous evolution. As an example, citizens may explicitly express their need to improve some quality requirement they expect from a software system at run-time.

# 3. Generic Software Engineering Research & Innovation Challenges

NESSI envisions the following generic, cross-cutting research and innovation challenges to be addressed towards achieving the Software Continuum. They are clustered into challenges related to (1) operations and adaptation, (2) development and evolution, (3) quality assurance and user experience, and (4) non-technical concerns.

## 3.1. Operations and Adaptation

**Run-time adaptation to "unknown unknowns".** Current adaptive software systems are able to react at run-time based on a pre-defined set of adaptation strategies, sometimes called the "known unknowns". Cloud platforms, service execution models, dynamic product lines, and code generation techniques are means to achieve such kinds of adaptation. However, it is necessary to provide mechanisms to handle the "unknown unknowns". This means incorporating at runtime new strategies to adapt systems to new and emerging requirements, unforeseen interactions with other systems or new and changing contexts. Novel solutions will enable higher level adaptation (e.g., adapting the adaptation model). This is especially critical in those systems that are continuously changing in composition and behaviour, such as Systems of Systems (SoS), which are dynamically composed by other systems. ***Justification for generic / centralized nature:*** The need for dealing with "unknown unknowns" is emerging in many technology and application domains. Examples for application domains include smart factories or smart cities. Examples for technology domains include CPS and the IoT, as well as fog computing and context-aware networking.

Developing fundamental principles for handling this uncertainty will have a profound impact across all these domains on how we build future software systems.

**Cooperative automated software.** Cooperation among automated software components or agents is required to handle the complexity and scale of large systems or applications. Due to their heterogeneity and loose coupling, the cooperation between software entities (multi-layer / cross-domain), with the users or with the external context must be based on high level semantics and policies. Basic self-* behaviours (such as resilience or automatic load balancing) can be implemented in the infrastructure and middleware. However, higher-level services and interactions with users currently lack the context adaptation, meta-models, semantics and heuristics necessary to implement true self-* capabilities of higher order (e.g., intentional programming, policies for societies of software agents). ***Justification for generic / centralized nature:*** With the increasing convergence of information and embedded systems (such as evident in CPS, but also future business processes, such as Industry 4.0), the capability of software to cooperate with other systems and humans in their context will become increasingly important. To cooperate, systems need to share a minimum amount of information and follow compatible principles, which can be delivered by implementing these principles as generic, foundational software engineering techniques and tools. Examples include explicitly modelling the systems' context by means of computational models.

**Self-managing fog and edge networks.** The proliferation of smart devices and sensors connected to the network demands fundamental transformations in whole ICT ecosystems. Such devices need to be configured and maintained after they are deployed. Managing networks of a very large number of heterogeneous devices, where each of which may run a multitude of services is challenging and complex. Advanced management of physical resources, including novel techniques for proper virtualisation and containerization of the fog nodes and the leaf nodes of the edge, will be key. In addition, new services for dynamic deployment and provisioning in the fog and edge networks (e.g., trusted support of cloud bursting) are required, as are new development paradigms and tools (e.g., DevOps enablement down to the remote devices). Recent advances on proper abstractions for largely heterogeneous tiny devices, container technologies that are able to run on smaller devices, as well as DevOps methods and techniques, are promising baselines. ***Justification for generic / centralized nature:*** Fog and edge will cover a wide range of technology domains. From the large-scale data centres (traditional cloud), over routers and edge devices up to connected sensors and IoT devices. As such self-management techniques and tools need to work along the whole compute continuum and thus require certain generic, underlying principles.

## 3.2. Development and Evolution

**Programming quality characteristics.** For some services and some users, the quality (non-functional) characteristics of a service (such as dependability, security, privacy…) may be more important than the direct functions it provides. Citizen programming should go further than simply describing the functional behaviour of a service, it should also facilitate the automatic implementation of quality requirements by the software generation chain up to the target platforms and infrastructures. Extension of aspect-oriented programming, offering dedicated APIs for software infrastructures, and model based tools may be good starting points, such are techniques for expressing service-level agreements and quality contracts. However, all these need to be raised to the problem space level, such that citizens may express their quality expectations in human terms. ***Justification for generic / centralized***

***nature:*** Enabling the citizens to "program" their quality requirements constitutes a technology-agnostic, new way of developing software.

**User development kit (UDK).** Going from Software Development Kits (SDKs), used by experienced software developers, to User Development Kits (UDKs) for citizens, will provide the framework and standards for ensuring the world is connectable. UDKs are for citizens what SDKs, framework architectures and patterns are for software engineers. Initial exemplars of UDKs exist; helping create games with Flappy birds, being a good example. ***Justification for generic / centralized nature:*** Creating the new notion of UDK constitutes a technology-agnostic, new contribution to software engineering.

**Collaborative software development.** Human beings are social and are becoming more and so in the virtual world. Increasingly, people will interact over networks, some of which are created by collaboration among citizens. Software will be the engine of innovation and new generation cloud platforms will provide software creation services where people will be enabled to work together using new generation collaboration tools. Future collaborative software tools will leverage intentional programming techniques, as well as the advances in cognitive programming for enabling people to create new software services, innovate and change the way we live, support new economic models and, finally, contribute create real open innovation. ***Justification for generic / centralized nature:*** Collaboration in software engineering is rather independent of the application or technology domain of the to-be-developed software.

## 3.3. User Experience and Quality Assurance

**Quality guarantees for self-learning systems.** Fully automated software will evolve and morph during run-time due to learning and cognition, without the explicit control of software engineers. This raises questions about the resulting overall behaviour of the system (of systems). Also, how to intervene in a fully automated system of system in case something goes wrong? Techniques that ensure that such software will always work within safe operational boundaries (e.g., in safety-critical situations, involving the IoT/CPS) are required. Current verification and validation techniques work during design time or exploit pre-defined strategies to anticipate how systems may adapt and evolve during run-time. Due to learning and cognition, such anticipation of run-time behaviour is no longer possible and requires a different approach towards quality assurance. Advances in learning techniques (such as deep learning) combined with more powerful means to deploy verification tools (such as verification in the cloud) will provide the required means to address this challenge. ***Justification for generic / centralized nature:*** The use of self-learning will increase across many different domains, as it represents one promising approach to deal with the "unknown unknowns" (see above). As such, similar to the challenges mentioned in Section 3.1, ensuring the quality of such learning systems will represent generic, cross-cutting software engineering knowledge.

**User-transparent quality assurance.** Software quality assurance is a key activity to ensure that software will not lead to unwanted side-effects or exhibit dangerous bugs. In particular in the case of citizen software, quality assurance needs to be performed in a transparent way, i.e., without blocking the creativity of the users. Current quality assurance techniques (such as code-coverage testing) are mainly targeted at professional software engineers and require explicit knowledge of how to use these techniques. Advances in online testing and run-time verification, as well as big data analytics for software repositories provide novels opportunities for automating the quality assurance process and thus making it transparent to the software developers. ***Justification for generic / centralized nature:***

Complementing citizen "programming" with user-transparent quality assurance contributes to the technology-agnostic, new way of developing software.

**Quality of experience for interacting services.** Humans will judge service quality based on the entirety of their experience, including the usability, functionality and operability of a service, across the whole chain / network of interacting services and systems. Quality of experience considerations are currently focused on single services rather than a chain / network of services. Applying Design Thinking to software-based services combined with Big Data analytics may help to collect information about usage, functions, and operations of services, thereby addressing this challenge. ***Justification for generic / centralized nature:*** The chain of interacting services in the context of the Software Continuum may cross many technology (cloud, edge, IoT) and application domains. As such, cross-cutting software engineering techniques and tools for assessing the quality of experience among those interacting services are required.

## 3.4. Non-technical Concerns

The rapid innovations towards a software-defined world presents challenges for citizens in terms of end-user skills and awareness. However, it is likely to be equally challenging for public and private sector organizations seeking to deliver the societal and economic value impacts expected from such innovations. The rapid innovation in the realm of software and services have not been matched by similar advancements regarding how to systematically achieve economic and social value from them. For example, over half of all large-scale technology deployments regularly fail to deliver the value and innovation expected of them[2].

With IDC estimating annual worldwide IT spending at around $2 trillion, such failures represent a significant loss of value, both economically and societally. While investigating the innovation occurring within software and services is needed, there is a risk such innovations may not see their incorporation in mainstream service delivery (by service-provider organisations) or adoption/uptake (by private citizens/end-users). Wider aspects associated with both of these groups needs to be considered in tandem. For example, security technologies has gained momentum in research and development, with the EC funding many initiatives to enhance the importance of the topic, such as the SOURCE network of excellence on societal security. More coordinated research and development on the wider aspects of innovations emanating from software and services realm is required; e.g., addressing ethical and privacy implications resulting from such innovations, addressing end-user skills proficiency, designing optimal organisational structure/capabilities to deliver the value from such innovations, optimal business models associated with service and software innovation delivery, etc.

To strengthen fostering innovation, skill building should also be built from the understanding the potential of software for creativity and thus learn from and collaborate with social sciences and humanities[3]. One important angle of such curricula may also include generalising work on open source projects during education, constituting a win-win situation for the "students" and the community. The long term impact of software relies on educating our future generation. In many respects, Europe has been foresighted in taking on ICT as part of programmes, e.g., in

[2] See British Computer Society, The challenges of complex IT projects. The report of a working group from the Royal Academy of Engineering and the British Computer Society. London: The Royal Academy of Engineering, 2012; and Bloch, M., Blumberg, S., and Laartz, J., Delivering large-scale IT projects on time, on budget, and on value. McKinsey & Company, 2012.

[3] See "Software engineering: Key enabler for innovation," NESSI White Paper, July, 2014. (http://www.nessi-europe.eu/Files/Private/NESSI_SE_WhitePaper-FINAL.pdf)

both primary and secondary schools. However, an understanding of software goes far beyond goals such as a digitally competent population and digital divide (cf. the Digital Agenda for Europe). The European education and school system would significantly benefit from strengthening its ICT training**,** on all levels and in particular in software. Designing and programming software as a basic skill will profit individuals, businesses/industry and society. In a sense, software literacy must become a key literacy skill to make Europe innovative and competitive and thus serve as a primary foundation for the job creation of this century. Combined with the aforementioned technology advancements in citizen software, this skill building will truly enable citizens to contribute to the value creation chain in the Digital Information Society and Economy 2.0.

# 4. Conclusions

Leveraging the opportunity the Digital Information Society and Economy 2.0, will significantly depend on our ability to create, manage and evolve software. The NESSI vision of the Software Continuum brings together two main streams for software creation, management and evolution: Self-managing Systems and Citizen Software. To deliver this vision, NESSI recommends starting to address key research and innovation challenges as part of the ICT Work Programme 2018 – 2019.

The challenges are relevant to many different application domains (including smart cities, smart factories, transport and logistics, agrifood, and so forth) and cover a broad range of technology domains (including IoT, CPS, Cloud, Fog, etc.). Yet, to avoid replicating investments and research effort, NESSI advocates setting up centralized activities on generic software engineering principles, techniques and tools, which may feed complementary more domain specific activities.