Innovation Potential of Software Technologies in the context of Horizon 2020

Lero THE IRISH SOFTWARE RESEARCH CENTRE

European Commission

Mike Hinchey

Director, Lero-the Irish Software Research Centre and Professor of Software Engineering, University of Limerick, Ireland

Brian Fitzgerald

Chief Scientist, Lero-the Irish Software Research Centre and Frederick A. Krehbiel II Chair in Innovation in Business and Technology, University of Limerick, Ireland

Brian Donnellan

Principal Investigator, Lero-the Irish Software Research Centre and Professor of Information Systems Innovation, Maynooth University, Ireland

Tiziana Margaria

Principal Investigator, Lero - the Irish Software Research Centre and Professor of Software Systems, University of Limerick, Ireland

---

23 September 2016
Prepared for
European Commission
Directorate-General of Communications Networks, Content and Technology (DG CONNECT)
Unit E.2 – Software & Services, Cloud

Table of Contents

Executive Summary

Software has become pervasive and increasingly complex. Many non-software products and services, from healthcare to transport, education to business, finance to energy, depend on reliable, high-quality software.

The increasing demand for software is fueled by the increasing capability of software to perform tasks that were previously accomplished through some form of hardware. Also evident is the move beyond Internet of Things (IoT) to Systems of Systems where the sensors and sources of data are fully integrated into web-enabled systems capable of utilizing machine learning techniques to offer real-time data analytics with the ultimate goal of enabling societal benefits.

However, many problems continue to exist in software projects, which can be attributed to poor preparation, poor project management, insufficient understanding, and insufficient resources. We have reached a second software crisis (Software Crisis 2.0).

One issue is the massive increase in the volume and quality of software required to fuel the demand in new domains where software has not been always of primary significance. The software required for domains such as health and medicine cannot be "more of the same": in the face of a bustling landscape of apps and applications for patients and professionals alike, most of the designers/developers do not have a technical background, and in the larger applications, such as Hospital Information Systems and similar, silos still dominate, making integration costly and difficult.

This situation is replicated across several business domains as the transformation to software has been taking place for quite some time. **We envisage a similar evolution to occur in a whole range of industries**, most notably in those industries traditionally associated with the production of physical goods or devices, where software has been a major source of innovation.

Innovation in software development is not fundamentally different from other types of innovations but certain traits set it apart along several dimensions of the technological and economic decision space: low capital investment intensity, continuous user contribution, and frequent releases and updates.

Companies in a range of industries are using software to achieve innovation, as software costs much less in material terms and quality software enables frequent and speedy change.

A number of domains relevant to software research are showing promise with great potential for innovation in areas such as financial services, smart agriculture and food, media of various formats, retail and transportation.

The Open Source Software (OSS) phenomenon has certainly transformed the traditional proprietary software industry in relation to how software is sourced and developed giving rise to globally distributed software. However, in the past decade, the proprietary software industry has also contributed in stimulating the evolution of open source software as many OSS products stem from both commercial and community participants operating in a complex symbiotic ecosystem.

In their Digital Agenda (www.ec.europa.eu), European commissioners listed 4 reasons for promoting Open Data initiatives, including potential economic gains from new product and service development, addressing societal changes, fostering citizens participation, and improving internal efficiency. Similar reasons apply in supporting the Digital Single Marking, Digitising Industry, Open Science and other initiatives. However, for Open Data to become valuable

there needs to be an integrated process that supports the entire data lifecycle management: from raw data collection and characterization with respect to quality, provenance, and legal usability, to its publication and accessibility as information via adequate safe and secure services, to a rich platform of analysis, aggregation, presentation and visualization in ways that make it useful for users to interpret as information.

It is essential that continued software research helps to raise the practice of software and system development to a fully–fledged professional discipline (similar to the other sciences and engineering) rather than the "craft" (reliant on a limited pool of talented and educated professionals, and a much larger pool of software development "immigrants", with limited training in coding, or even less) that it is today. Without such a rise in qualification profile and competence expectations, innovation will be stifled and many industries currently reliant on software for innovation will fail to meet their potential

Highest quality and increased productivity are particularly essential in "high-tech" Europe, where loss of innovation will mean loss of competitiveness and ultimately economic lost opportunity.

This migration requires that the Commission funds research that will:

Develop new models and paradigms to enable the next generation of software development and higher level languages adequate for both target tiers: **subject matter experts** who participate in the innovation and design, but do not code, and **skilled IT professionals** able to create innovation within the IT systems and their production lifecycle;

Enable scalable, tool-supported, and efficient, development methods that address specific domains, industries, organisations and processes;

Enable active participation by customers in the software ecosystem and make software development customer-led (need-"pull" rather than technology "push"), while ensuring security and privacy of personal information, particularly in light of Cloud computing technologies, open source software, open data, and social media and other platforms;

Enable (via tools, technologies, languages, and paradigms) speedy and cost-effective development of highly-reliable software, able to express *physical*, *cyber* and *social* design objectives simultaneously, but that is also able to evolve without loss of reliability nor prohibitive cost.

EU-funded programmes have resulted in the creation of a very large amount of software. However, much of this software does not survive beyond the life of the project being *funded.* While *releasing such software as open source was seen as a way of ensuring better longevity* and a positive approach, this licensing per se has little effect without a development community sufficiently socialized to work together in the long term towards common goals.

Our recommendation is that to facilitate innovation, further EU research investment should emphasize:

- Investing in open science initiatives, such as the foundation of non-profit organizations that can facilitate open access publications.
- Fostering Open Collaboration and the benefits that accrue from it.
- Supporting (and possibly mandating) Open Data throughout EU-funded research.

Similarly, while there are many topics that are not within DG CONNECT's remit, it should continue to support a wide-ranging software and services research programme, such as its remit allows. However, while keeping this broad range, it would be worth emphasizing the criticality of software and its need to evolve. That is why we recommend a research programme that emphasizes the area of Evolving Critical Systems, supporting the development of software-intensive systems that are reliable and retain their reliability as they evolve, in support of innovation in European industry, fostering growth and leading to social inclusion.

1 Introduction

There are few areas of modern life in which software is not an important (though often invisible) component.

Millions of lines of software are used every day in many diverse areas from Agriculture to Automotive Systems (including, but not limited to, self-driving cars), Entertainment to Disaster Recovery, FinTech (Finance) to Medicine (both research and treatment).

Software is also the major source of innovation and advancement in a large number of areas that are current "hot topics". Topics at or near the peak of inflated expectations, or above the trough of disillusionment in the Gartner ICT Hype Report 2014, see Figure 1.

THESE INCLUDE:

- Internet of Things;
- Data Science;
- Big Data;
- Gamification;
- Hybrid Cloud Computing;
- 3D Printing.

To these we can add topics such as *Cyber-Physical Systems* (CPS), where Cyber-based systems (i.e., software) interact with the physical world by means of various sensors and actuators. They include wearable devices, medical devices, and interfaces to social media, implying the growing importance of so-called *Cyber-Physical-Social* systems such as wearable computing, smart-cities, smart-grids, smart-*, and many other areas in health and communications, all inherently dependent on large scale and high assurance software.

In some of these areas, other jurisdictions (such as USA) are taking the lead. However, there are areas where Europe can still potentially leap ahead.

2 The Software Crisis

2.1 Background

The software in our lives is increasingly complex; its interaction with the real world means that its requirements are in a state of constant change (Lehman & Fernández-Ramil, 2006; Hinchey and Coyle, 2009). Many non-software products and services, from healthcare to transport, education to business, finance to energy, depend on reliable, high-quality software.

Software engineering is the discipline that applies systematic, rigorous engineering principles to the design and development of software, much as civil and mechanical engineering do to the construction of buildings and machines. Software engineering improves the quality, reliability and predictability of software systems by generating knowledge, methods, tools, and development processes that both facilitate and improve the product: software. These qualities are essential wherever software failure might lead to significant safety, security, or economic losses.

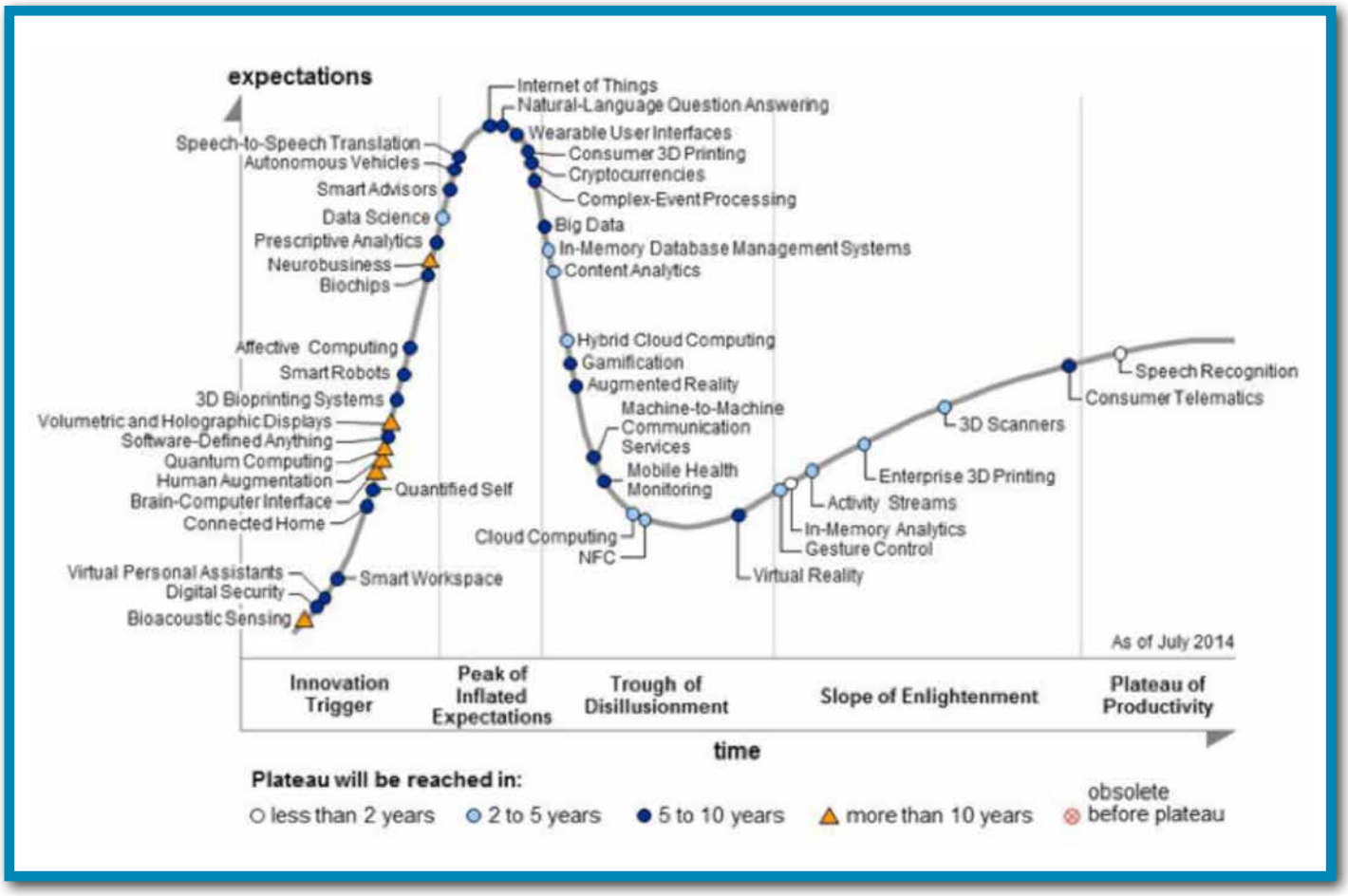

Fig. 1: Gartner ICT "Hype" Chart as at July 2014
(available via: http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp)

	Than 5 Years Ago	Than 10 Years Ago
Significantly More Failures	27%	17%
Somewhat More Failures	21%	29%
No Change	11%	23%
Somewhat Fewer Changes	19%	23%
Significantly Fewer Failures	22%	8%

Fig. 2: Project failures compared to 5 and 10 years ago (Standish CHAOS Report, 2015)

Change management is a longstanding issue in software systems: software systems need to be modified in response to changes in system requirements and in their operational environment). Such modification may involve the addition of new functionality, the adjustment of existing functions, or the wholesale replacement of entire systems. All such change is fraught with uncertainty: software projects involving change frequently fail to meet requirements, run over time and budget, or are abandoned (Rajlich and Bennett, 2000; Hinchey and Coyle, 2009).

Much of the problems with software projects can be attributed to poor preparation (requirements misunderstood or incomplete), poor project management, insufficient understanding, and insufficient resources. The Standish CHAOS Report (2015) estimates that 31.1% of software projects will be cancelled before completion, and that 52.7% of those that are completed will cost 189% of their original estimates. The same report indicates that management believe that more projects fail now than 5 or 10 years ago despite significant advances in a wide range of technologies (see Figure 2).

2.2 Software Crisis 2.0

The increasing demand for software is fueled by the increasing capability of software to perform tasks that were previously accomplished through some form of hardware. This is evident in developments such as software-defined networking (Kirkpatrick, 2013), software defined infrastructure (Fitzgerald et al., 2015), software defined data centers (Dell, 2015), right through to the concept of the software-defined enterprise, which has enough intelligence to automate all decision-making and business processes ("Enterprise Physics" – Margaria and Steffen, 2008). Indeed, we are at the point of **software-defined*** (where * can refer to networking, infrastructure, data center, enterprise, etc.).

This is also evident in the move beyond Internet of Things (IoT) to Systems of Systems where the sensors and sources of data, such as household-appliances, are fully integrated into web-enabled systems capable of utilizing machine learning techniques to offer real-time data analytics on the morass of acquired raw data, with the ultimate goal of enabling societal benefits for citizens through the provision of useful and precisely customized information – the quantified self, for example. In this era of '**Big Data**', Software is becoming the unifier for value creation, even for companies that sell physical products. The reason why software is preferred over hardware is due to *its ability to be changed anytime*.

It is clear that a massive increase in the volume and quality of software being produced is required to address the emerging initiatives named above and many other innovations. This creates what Fitzgerald (2012) describes as a Software Crisis 2.0 bottleneck[1]. The initial software crisis referred to the basic problems of time, cost and quality, first identified in 1968: already then, software took too long to develop, cost more than budgeted, and did not meet user expectations when eventually delivered. Over the decades, several initiatives have sought to address this crisis, e.g., the waterfall life-cycle, the structured approach, software product lines, software patterns, agile methods, model-driven development However, none has succeeded in delivering an order of magnitude increase in software development productivity. Rather as the well-known and often cited Standish report[2] suggests, software project failures are quite often the norm. However, it is true to say that model-driven development, software product lines, and agile methods, amongst others, are delivering significant improvements, with widespread industry

1. *The first Software Crisis was identified at a 1968 NATO conference at which the term "software engineering" was first coined. The phrase is more commonly associated with a highly-cited article in Scientific American by W.W. Gibbs.*
2. *https://www.projectsmart.co.uk/white-papers/chaos-report.pdf*

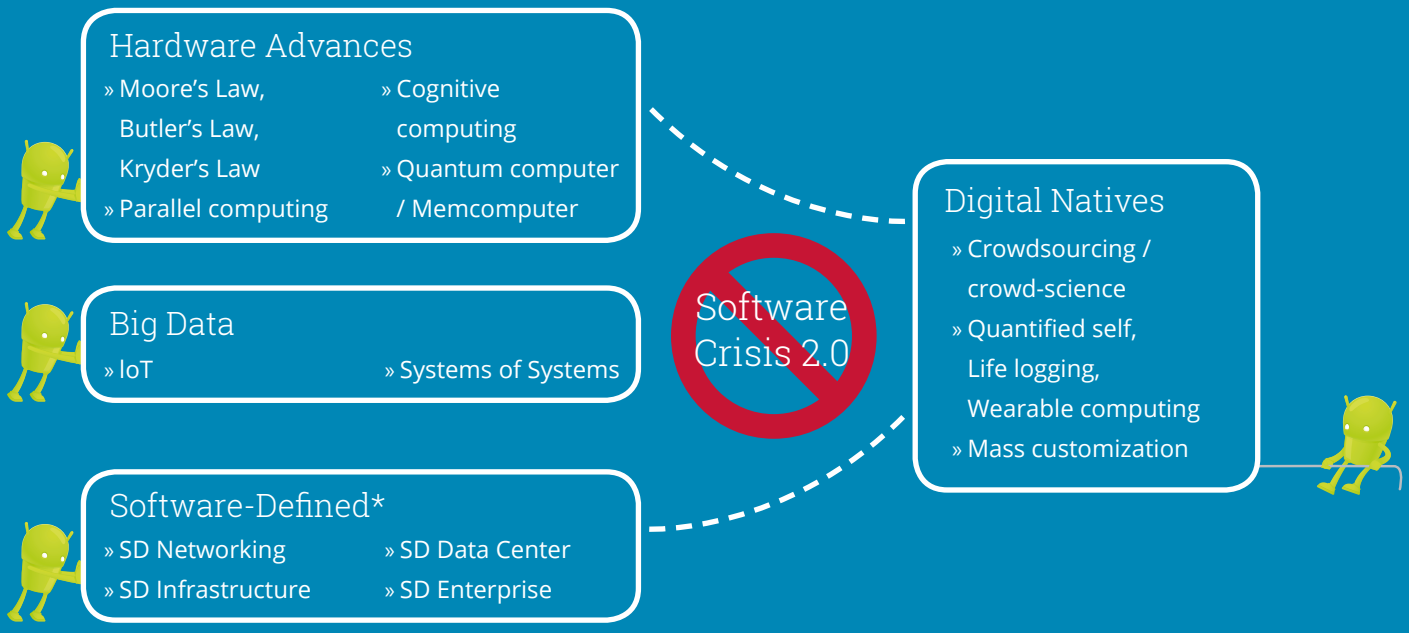

Fig 3.: Software Crisis 2.0

acceptance, having passed the stage of being exclusively research topics.

This needs to be examined in the context of advances in other ICT areas – **hardware advances**, for example. In the hardware domain, the well-known Moore's Law in relation to integrated circuits is paralleled by other exponential improvements in data transmission and storage capacity, as exemplified by Butters' Law and Kryder's Law respectively. In the areas of parallel processing and multicore computing, however hardware advances require significant additional technical expertise at the software level in order to be leveraged successfully.

An interesting distinction has been drawn between "digital immigrants" – those who began using digital technology at some stage during their adult lives – and "digital natives" – those immersed in the world of technology since birth, developing a natural fluency for technology (Prensky, 2001). By the age of 20, digital natives will have spent 20,000 hours online (Valkenburg and Peter, 2008) and can cope with, and indeed even welcome, an abundance of information (Vodanovich et al., 2010). Digital native consumers represent a significant "pull" factor in seeking to take advantage of the opportunities afforded by advances in processing power and increased availability of data. The advent of wearable computing fuels big data and has led to initiatives such as life-logging and the quantified-self. With such initiatives individuals can collect data about all aspects of their daily lives – diet, health, recreation, mood states, performance – in some cases recording a terabyte of data per annum (Gurrin et al., 2014).

How to optimally leverage digital natives as IT and software producers is however still unclear. What we see is that the traditional education needs to evolve and accommodate the changed abilities and aptitudes of the native generation.

The paradoxical success of the open source software phenomenon has led to a broader interest in crowd science or citizen science as a generalizable collaborative model of problem analysis and solving. Notable areas of success are user-led innovation, co-creation of value, and high profile crowdsourcing of solutions for solving complex R&D problems in organizations such as NASA, Eli Lilly and Du Pont, which provides real testimony to the potential of the digital native.

Mass customization has been succinctly defined as "producing goods and services to meet individual customer's needs with near mass production efficiency" (Tseng & Jiao, 2001). While not a new concept, it resonates well with catering to the personal needs of the digital natives. Also, it is now typically delivered through software-mediated configurability to meet individual customer needs. The concept of automated personalization is linked to the desired benefits of big data.

These various "push" and "pull" factors are presented in Figure 3 below. It is clear that a massive increase in the volume and quality of software being produced are required to address these emerging needs and opportunities. Together, they define a Software Crisis 2.0 bottleneck as illustrated in Fig. 3.

2.3 New Skill-sets Required for Software Designers and Developers

There are two dimensions to this crisis. One is the massive increase in the volume and quality of software required to fuel the demand in new domains where software has not been always of primary significance – medicine and healthcare for example,

where terabytes of raw data need to be analyzed to provide useful actionable insights. Here, the software required for these domains cannot be "more of the same": in the face of a bustling landscape of apps and applications for patients and professionals alike, most of the designers/developers do not have a technical background, and in the larger applications, such as Hospital Information Systems and similar, silos still dominate, making integration costly and difficult. Data is difficult to access, interfaces are complex and reflect legacy constraints, and most of the processes and workflows are buried in (often proprietary) software layers well below the wide accessibility needed to enforce change. Even if no major IT paradigm change requires developers to possess fundamentally new skills and techniques, service orientation, variant management, flexible integration, and quick, safe and secure workflow adaptation require a new generation of software designers that are comfortable with the high speed turnaround needed in this experimental software production. Ensuring that the co-creation works, with skilled and empowered users that co-design complex applications, requires a new generation of *descriptive means* likely closer to models than to code, *verification means* on these descriptions, with concepts of correctness and compliance that make sense to such broader audience as well as to the technology experts – a new take on knowledge management (Margaria and Steffen, 2010) much simpler than the semantic web technology currently available, and *design-development envir*onments that support a much more gradual and traceable transition from the "what" to the technical "how" than the model driven design approaches currently available (Margaria and Steffen, 2005). In terms of the software development industry, this challenges the status quo, and whomever leads the disruption is likely to shape the next generation of IT for all of society.

In the overall backdrop to this software bottleneck, however, it is worth bearing in mind that estimates suggest the population of professional software engineers worldwide to comprise no more than 500,000 people (Grier, 2015). Clearly, there are more software development practitioners in the world, and development resources may even be boosted by a general willingness for additional people to get involved based on a crowdsourcing model. However, the skills required in this brave new world are not those possessed by the average software developer.

The second dimension requires software development practitioners to acquire fundamentally different new skills.

Parallel processing on multicore architectures, for example, poses a number of fundamental challenges. The traditional programming paradigm with run-time task allocation and scheduling lets the operating system allocate tasks to processors and take care of scheduling and load balancing. In a multicore architecture, these decisions can be made at design-time or compile-time and developers need to design program threads accordingly. This is good, as

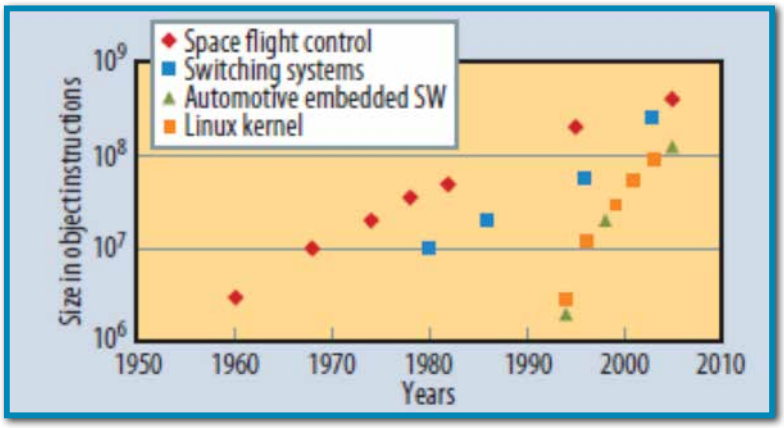

Fig.4: The increasing size of software systems (Jones and Ebert, 2009).

whatever can be checked at design time (also using formal methods) avoids runtime issues and bottlenecks. However, analysis, design, and debugging of parallel systems are significantly more challenging, and an optimization/tuning phase is necessary. While parallelization is successful for specific platforms and application profiles, it falls short of general and easy applicability. In the analysis phase, for example, the question is what code is worth parallelizing (the easy parts, like loops, or the frequently executed parts) and what parallelization is efficiently and safely feasible, given that hidden dependencies and race conditions may have horrible side effects. In the design phase, issues such as methods of threading and decomposition need to be addressed. In the validation and verification phase, handling data races and deadlocks and implementing thread synchronization accordingly are in focus. The optimization/ tuning phase considers performance issues such as the amount of code parallelism, and whether performance benefits can be achieved as the number of processors increases.

Figure 4 shows an increase in the size of software based on a 2009 article (Ebert and Jones, 2009). While one would expect the growth in switching systems (with the advent of software-defined networking, the internet, and mobile phone technology) and in space flight control, what is more interesting is the growing in Automotive embedded software and the Linux kernel.

Automotive software scarcely existed prior to the mid-1990s but as is pointed out below has now grown phenomenally. The growth in the Linux kernel can be explained by the participation of thousands of (often amateur) developers in this popular Open Source project.

Design lies at the heart of the software innovator of the future. The emphasis on design may require new skill sets for the software innovation team—which may include graphic designers, user experience engineers, cultural anthropologists, and behavioural psychologists. Designing engaging solutions requires creative talent; creativity is also critical in ideation.

The world's largest bookseller (Amazon), the largest video service by number of subscribers (Netflix), and the largest music companies (iTunes, Spotify, Pandora) are all now effectively software companies (in that they are fully reliant on software in order to provide their services), and the traditional companies which they are overtaking are reverting to becoming software-intensive companies.

Design lies at the heart of the software innovator of the future. The emphasis on design may require new skill sets for the software innovation team—which may include graphic designers, user experience engineers, cultural anthropologists, and behavioural psychologists. Designing engaging solutions requires creative talent; creativity is also critical in ideation—helping to create a vision of reimagined work, or to develop disruptive technologies deployed via storyboards, user journeys, wire frames, or persona maps. Some organizations are already going so far as to hire science fiction writers to help imagine and explain moonshot thinking. Figure 5 below shows the STEM occupations in high demand: 2012–2022 (projected) while Figure 6 suggests that to enable innovation in the future, that the conventional, traditional skills associated with software engineering will need to be augmented by cross disciplinary skills.

Fig. 5: IT Skills Projections in the USA

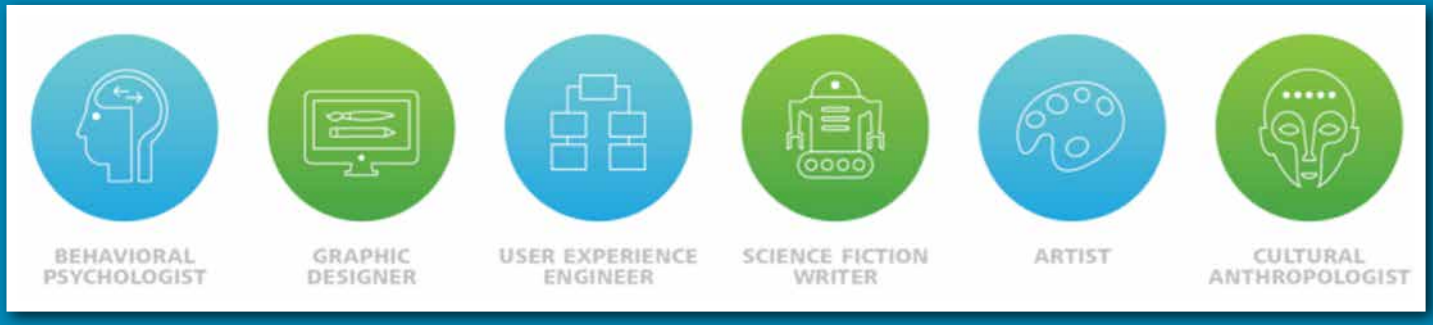

Fig. 6 : Cross Disciplining Skills to support innovation

3 Software as an Innovator

"Our organization has become a software company. The problem is that our engineers haven't realized that yet!"

This is how the Vice President for Research of a major semiconductor manufacturing company, traditionally seen as the classic hardware company, characterized the context in which software solutions were replacing hardware in delivering his company's products.

This situation is replicated across several business domains as the transformation to software has been taking place for quite some time (Fitzgerald, 2016). The telecommunications industry began the move to *softwareization* in the 1970s with the introduction of computerized switches, effectively inventing Service Oriented Computing in the 1980s with the Intelligent Network ITU standard. Currently, the (mobile) telephony market is heavily software-focused, with successes like Skype and the like that disrupted the traditional business model and technology stack. The automotive industry has very noticeably been moving towards softwareization since the 1960s—today, 80 to 90 per cent of innovations in the automotive industry are enabled by software (Mossinger, 2010; Swedsoft, 2010). This is evidenced in the dramatic proportional increase in the numbers of software engineers employed in the automotive sector versus those in traditional engineering roles. As a striking example of the growing importance of software in the automotive industry, illustrated graphically in Figure 5, while in 1978, a paper stack printout of the lines of code in a car would have been about 12 centimetres high, by 1995 this was already a three-metre high stack, and by 2005 50 metres tall. By 2013, the printout had grown to a 150 metres height. The estimate for 2020 is a staggering 830 metres tall, higher than the Burj Khalifa—currently the tallest man-made structure in the world (Schneider, 2015).

This is supported by a report by Siemens (2014), estimating that software had expanded from about 100 lines of code in the 1970s to as much as 10 million lines. It points out that *"New functions will be integrated not in the form of control devices, but as software. The third step, finally, would be a further virtualization of the necessary total system of hardware and software (the hardware/software stack) into a service-oriented architecture. The underlying execution platform composed of control devices and busses, would entirely virtualized by middleware."* **Software middleware for control devices meanwhile is commercially available, see e.g. TwinCAT by Beckhoff.**

We envisage a similar evolution to occur in a whole range of industries, most notably in those industries traditionally associated with the production of physical goods or devices.

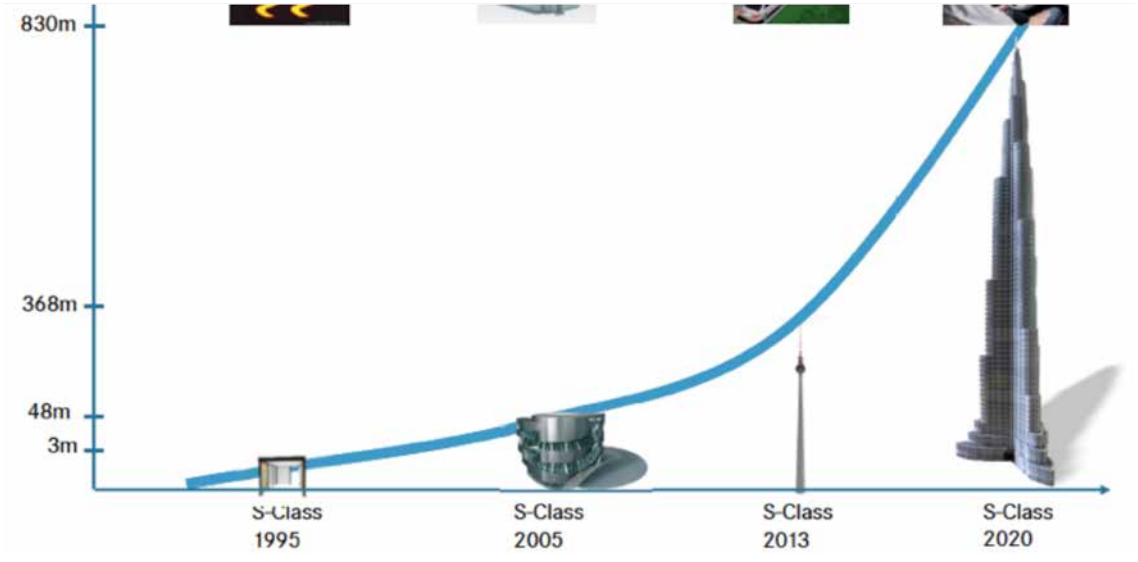

Fig. 7: Height of Software Printout in Mercedes S-Class *(Schneider, 2015)*

3.1 Software Product and Process Innovation

Innovation is the implementation of an idea or an invention which leads to improving and perfecting a product, a method, a theory or a service with the sole purpose of accomplishing, at a higher standard, the objectives they were originally designed for. Innovation may be the subject of an entire project or may occur in a certain fraction of a project.

A basic distinction can be made between product innovations as observed in the development of a useful new software application and process innovations such as the introduction of a new software development methodology[3] (see Figure 8).

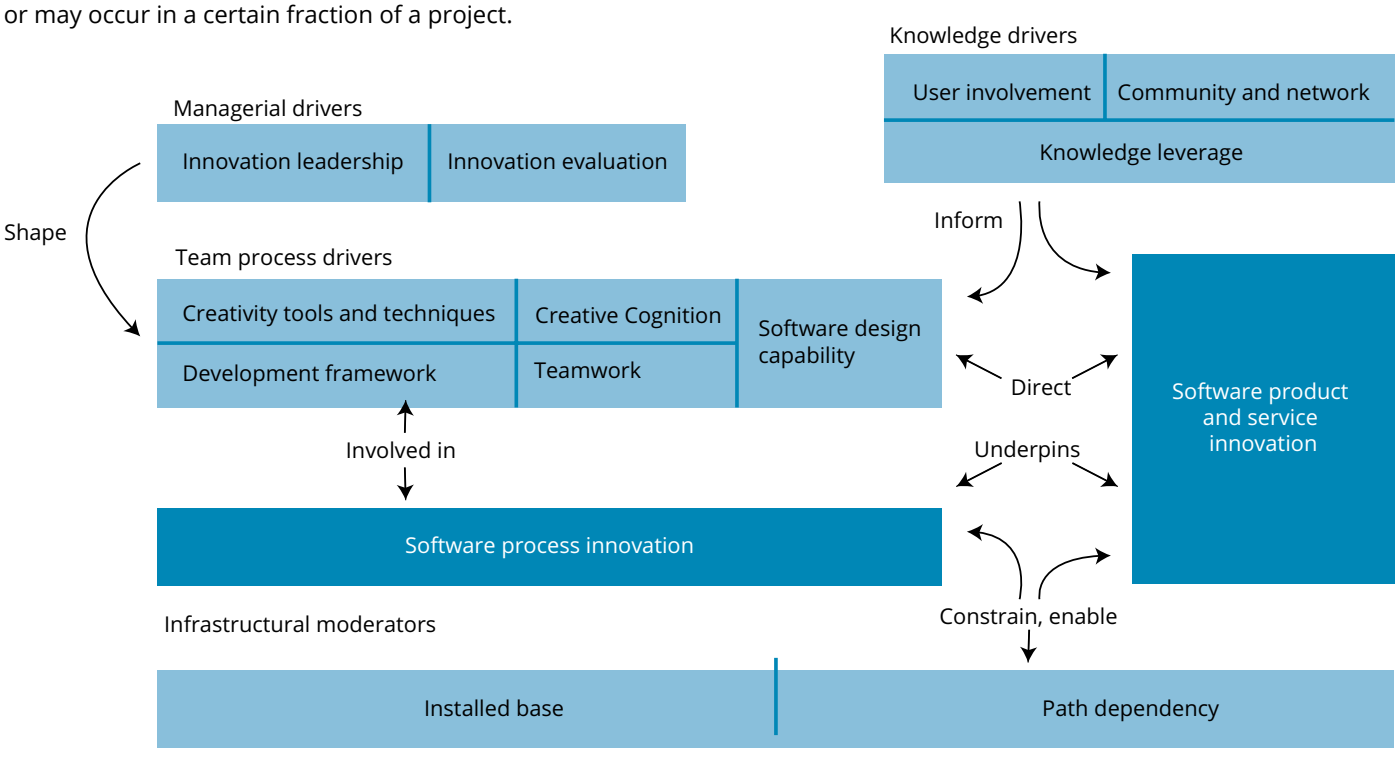

Fig. 8: Software Innovation Drivers

3. *Rose J., Furneaux B, Drivers and Outputs for Software Firms: Literature Review and Concept Development, Advances in Software Engineering Volume 2A, 2016*

The main drivers / influencers of each type of innovation have been identified below:

MANAGERIAL DRIVERS:

Through monitoring, control, and direction setting efforts, managers influence a wide range of important parameters that can have significant implications for software innovation. These parameters include resource allocations, work environments, strategic goals, and specification of the initiatives that are included in a project portfolio.

INNOVATION LEADERSHIP:

Innovation leadership has been identified as having a powerful influence on software innovation in relation to process innovation with innovation leadership being the second most prominent driver of software process

INNOVATION EVALUATION:

The ability to evaluate creativity and innovation in software engineering work is an important precursor to improvement. Evaluation takes the form of assessing the work environment, assessing the value of competing ideas during ideation, assessing new software product concepts, determining the value of process improvements and creativity support systems, and determining the value of the software services currently in use.

KNOWLEDGE DRIVERS:

A second group of drivers are knowledge-oriented factors that relate to the acquisition and leveraging of knowledge from internal and external stakeholders and the relationships that develop as part of these efforts.

KNOWLEDGE LEVERAGE:

Prior work suggests that knowledge plays a central role in many aspects of software innovation including creative requirements elicitation and understanding innovation opportunities.

COMMUNITY AND NETWORK:

Since knowledge creation and use are understood to be a social process, innovation researchers have tended to emphasize the importance of communities and networks to successful innovation.

USER INVOLVEMENT:

Though some innovations are driven by system developers and designers, research has increasingly stressed the role of users in software innovation. In particular, customers can play an important role in the commercialization of software inventions, assisting with customization, requirements elicitation, and early investment.

TEAM PROCESS DRIVERS:

Software is usually produced in teams that synthesize the creative ideas of team members and external knowledge into code that yields the product/service offerings of software development organizations.

CREATIVE COGNITION:

Creative cognition research aims to understand the creative state of mind and the creative acts of individuals, categorize different innovation styles, develop creative thinking, and foster creative talent in engineers. The basic premise is that the creativity of participants in a system development initiative can contribute significantly to the level of innovation generated by this initiative.

SOFTWARE DESIGN CAPABILITY:

Design capability encapsulates developer capacity to integrate customer understanding, market understanding, and technological advances into novel and useful product features

TEAMWORK:

Effective teamwork is considered an essential feature of innovative projects, contributing to team efficiency and the personal satisfaction of team members. The blend of experiences and competencies found in the composition of a team are also of fundamental importance to innovation.

INNOVATION TOOLS AND TECHNIQUES:

Software innovation is often assumed to benefit from access to a repertoire of suitable creativity techniques and support tools as well as situational knowledge of when to apply them.

DEVELOPMENT FRAMEWORK:

Development frameworks provide the foundation for an organization's approach to software development by offering support for processes, underlying assumptions, and work practice norms.

INFRASTRUCTURE MODERATORS:

In addition to the managerial knowledge, infrastructure determines the fundamental, unavoidable technological and social conditions in which innovative efforts are situated. These can include broadband availability and speed, microprocessor power, customer experience, and user computer literacy.

INSTALLED BASE:

Since the installed base evolves independent of a developer group, the timing of innovations can become crucial. If an innovation is too early, supporting infrastructure such as communication bandwidth and processing power.may not be widely available. If it is too late then the innovation has likely become evident to many competitors.

PATH DEPENDENCY:

Innovations typically grow out of existing software systems which are, themselves, part of the knowledge infrastructure. As a result, technological capabilities engender a form of path dependency that can have significant implications for the innovations that are achievable and may require major leadership interventions to change.

In software development projects, innovation occurs with an increased frequency. Innovation in software development is not fundamentally different from other types of innovations but certain traits set it apart along several dimensions of the technological and economic decision space:

A. **Low Capital Investment Intensity:** Innovation in the software development industry is not bound to sophisticated research laboratories or the prerogative of international scientific teams. A good idea, implemented correctly can be validated by online communities and quickly spread as a world acknowledged innovation.

B. **Continuous User contribution.** Users play an important and sometimes direct role in software innovation. User feedback is fundamental in any industry but with software development is very easy to use it at its full potential. Open source software encourages users to actively take part in the process of innovation implementation, but companies benefit increasingly from having users in the loop also beyond the open source domain, as software innovation can also be validated with the use of online user communities.

C. **Frequent Releases, updates, patches.** Software products are quickly evolving entities. They rapidly change during their entire existence thus providing the perfect environment for continuous innovation to take place. Some applications are even designed from the start to be launched progressively, module by module or in successive individual releases.

Other traits present in any industry are particularly beneficial to the fast paced and non-physical software sector.

3.2 The Increasing Importance of Software in Innovation Processes

The companies that in a range of industries are increasingly using software to deliver innovative, feature-rich products do so for very good reasons:

A. Innovation through software (as opposed to hardware) costs much less in material investment and stock: software is non-physical, needing no warehouse nor shipment as do physical goods.

B. Well maintained software, as the name indicates, is more flexible and changes can be made at a fraction of the time and cost, keeping the software "manufacturing" costs at bay.

C. Innovations delivered through software allow organizations to create more product variants at less cost to pursue new markets and new customers.

This trend is prevalent in many industries, but particularly in the automotive, aerospace and defense, industrial, medical device, and electronics and high tech sectors[4]:

A. Automotive: Automatic, intelligent braking systems on many high-end vehicles react in a potential crash situation even if the driver does not. Cars will require 200-300 million lines of software code in the near future. (Source: Frost & Sullivan)

B. Aerospace and Defense: Sensors in unmanned aircraft gather surveillance data for the US military. The Airbus A380 uses almost 1000 software components supplied by more than 40 system and software companies located on 3 different continents (Source: SITA)

C. Medical Devices: Software in implantable pacemakers monitors cardiac rhythms in heart patients. In the last 7 years, 500 medical device recalls were related to software defects and malfunctions (Source: FDA Survey).

D. Electronics and High Tech: From voice-activated texting to wireless internet access to privacy and security features — cell phones (and many other electronic devices) get their intelligence from software. (Source: Panasonic Corporation)

Speed of production and lines of code are however in sharp competition with **quality and assurance**. Therefore it is of vital importance to not just concentrate on the code itself and its quantity, but also on the properties it has, and the assurance levels we can attach to it in terms of predictability and controllability of its behavior, especially when it acts in unpredictable and uncontrollable environments. Non-interference, safety, security, compliance to norms, standards, regulations, and guidelines are here of paramount importance. For this, traditional methods of (still largely manual) validation and testing are hopeless, because they do not scale.

» A leading medical device company has reduced compliance reporting efforts by 99% — from up to 36 person-weeks per FDA regulatory submission to mere minutes — speeding time-to-market and improving safety in product engineering. A pioneer in unmanned aircraft systems (UAS) for intelligence, surveillance, and reconnaissance was able to move from Capability Maturity Model Integration (CMMI, a software engineering process improvement approach designed to help organizations improve performance) Level 1 (i.e., unpredictable, reactive processes) to Level 3 (i.e., proactive processes customized for the organization) in 11 months by using a single automated solution to manage all core CMMI processes and artifacts. This company reached the 500,000 combat flight-hour milestone faster than any other UAS company.

» Although these companies represent a range of industries, sizes, and growth stages, they all share common elements: they innovate through software and they all use a single solution to automate product engineering throughout the product development lifecycle. They are leaders in their market niche and they sustain this edge by getting products to market quickly. These industry leaders have streamlined the embedded software development process and automated best practices across the product development lifecycle. By improving process efficiencies, these companies are able to focus on their core business — accelerating innovation and speeding new products to market -- which translates into larger market share, increased revenues, and higher profits.

4. "Software: Driving Innovation for Engineered Products", www.PTC.com

3.3 Software as an enabler of Future Digital Disruptions

The advent of mobile technology and social media has had a profound disruptive effort on retail and travel industries. While, by definition, it is hard to predict where disruptions may occur, there is a growing consensus that any sector that is based on a brokerage model will be vulnerable to disruption in the short or medium term[5]. For example the financial services and real estate sectors have an enormous potential for disruption. Software-based transactional systems have the potential to address the inefficiencies in the brokerage model. Currently the most topical examples of this type of software technology are crypto- currencies and crypto-equities. Software start-ups are emerging that are developing technology to allow investment in a company without ever using traditional money. These types of software innovation have the potential to disrupt the global economy and banking systems. It's still a nascent phenomenon but the potential for Internet-scale disruption that could change the way we transact business is becoming apparent.

The blockchain[6] has the potential to usher in a new era characterized by global payment systems, digital assets, decentralized governance, and even decentralized legal systems. It enables collective organizations and social institutions to become more fluid and promote greater participation, potentially transforming how corporate governance and democratic institutions operate. The technology could impact capital markets, by enabling everyday citizens to issue financial securities using only a few lines of code. Beyond these opportunities, the blockchain has the possibility to fundamentally change the way people organize their affairs. The technology can be used to create new software-based organizations referred to as decentralized organizations (DOs) and decentralized autonomous organizations (DAOs). These organizations can re-implement certain aspects of traditional corporate governance using software, enabling parties to obtain the benefits of formal corporate structures, while at the same time maintaining the flexibility and scale of informal online groups. These organizations also can be operated autonomously, without any human involvement. They can own, exchange, or trade resources and interact with other humans or machines, raising novel questions around traditional notions of legal personality, individual agency, and responsibility.

Software developers have quickly realized the potential for blockchain technology and have started to use it to create digital currencies, self-executing smart contracts, as well as cryptographic tokens that can represent property or ownership interest in emerging services. It is also being used to create: censorship-resistant communications and file sharing systems; decentralized domain name management systems (DNS); and fraud-resistant digital voting platforms.

5. *Digital Transformation Review, Capgemini Consulting, February 2015*
6. *Wright, Aaron and De Filippi, Primavera, Decentralized Blockchain Technology and the Rise of Lex Cryptographia (March 10, 2015). Available at SSRN: http://ssrn.com/abstract=2580664*

3.4 Potential Areas of Innovation relevant to Europe

A number of domains relevant to software research are showing promise, some with track-records in various European research centres and groups, others already actively being investigated in other jurisdictions.

3.4.1 Financial Services – APIs for Currency Transactions[7]

Precursors to modern Service-Oriented Computing, APIs provide code-level access to systems for programmers to develop applications on top of existing software without necessarily understanding the internal details of how the system in question is implemented. It is sufficient to know how it works. More importantly, however, service interfaces and Open APIs – those made available to external developers – dramatically reduce transaction costs and help to create innovative third-party applications and new markets. For example, cryptocurrencies are unique as they are effectively providing the first set of Open APIs for money. These APIs may assist in the creation of the sharing economy through allowing multiple people to co-sign, or pay for, an item that they wish to share ownership of. For example, a number of people could combine their keys to pay for a song or video that they all would co-own and share.

3.4.2 Smart Agriculture and Food[8]

Software is being applied in a broad variety of areas of farming including monitoring and management of crops and cattle, maintenance of farming equipment, and mapping of fields and other operational activities to optimize watering and irrigation, the sowing of seeds, etc. These solutions are becoming economically viable due to the reduced cost of tailor-made sensor solutions, the cheaper cost of storage and processing in cloud infrastructures and relatively cheap bandwidth (fixed wireless) that permits the transmission of data sets from fields across nations and regions.

3.4.3 Media[9]

The creation of aggregators in the media industry has produced websites or software systems that pull together different types of information and content for end users. These aggregators enable users to create their own bundles instead of relying on a company to do it for them. Rich content and user produced content are increasingly being targeted, with entire stacks of technology for the cloud (e.g., OpenStack) and for real-time media (e.g., WebRTC) increasingly going open source and attracting interest for industrial exploitation.

3.4.4 Retail[10]

Many retailers already have a large installed base of CCTV, or infrared cameras, originally installed to reduce the likelihood of theft. With the application of back-end software, however, it is now relatively easy for retailers to individually track customers as they move and stop around the store. By applying learning algorithms to collected data, retailers are able to redesign their store layouts in a manner that is more appealing to customers and group different products together to increase the possibility of sales. RetailNext, for example, has developed software that uses a store's existing security cameras to give managers all kinds of information about how consumers interact with the store. They can show exactly how many customers are in a store at a given time, which parts of the store they explore, which specific items customers spend more time perusing, and which they do not. They can combine this information with other variables like staffing levels, weather, product assortment and placement to determine their effects on sales. Mont Blanc has used RetailNext's services to improve its staffing levels and its product arrangement within its stores, increasing same-store sales by 20% in the process.

---

7. Industry Transformation – Horizon Scan: ICT & the Future of Financial Services, Ericsson Networked Society Lab
8. Industry Transformation – Horizon Scan: ICT & the Future of Agriculture, Ericsson Networked Society Lab
9. Industry Transformation – Horizon Scan: ICT & the Future of Media, Ericsson Networked Society Lab
10. Industry Transformation – Horizon Scan: ICT & the Future of Retail, Ericsson Networked Society Lab
11. Industry Transformation – Horizon Scan: ICT & the Future of Transport, Ericsson Networked Society Lab

3.4.5 Transport[11]

End-to-end automated software engineering solutions help focus software development resources on core product development and product innovation, leading to more competitive product lines and accelerated time-to-market with new features. For companies in a range of industries, adopting this type of solution has made a huge impact. One of the world's leading automotive companies adopted an automated end-to-end engineering solution to help manage the volume and velocity of engineering change driven from software. In this company, 90% of product changes are software-based. By deploying a single solution for the entire product development lifecycle, quality has improved, costly rework had been eliminated, regulatory reporting has been simplified, and requirements and change information is easily shared throughout the organization and with OEM partners. A leading supplier of in-car location and navigation services used a single product engineering solution to create an early warning system that finds and corrects issues before schedules, quality, or costs are impacted. As a result, the first time right statistic improved from 80% to 97%. Release predictability was also improved, providing the ability to deliver as promised on-time products to customers.

3.5 Open Source Software

The Open Source Software (OSS) phenomenon has certainly transformed the traditional proprietary software industry in relation to how software is sourced and developed (Fitzgerald, 2006) giving rise to software ecosystems such as Google's Android platform (and third-party apps) that are globally widespread.

However, in the past decade, the proprietary software industry has also contributed in stimulating the evolution of open source software as many OSS products stem from both commercial and community participants operating in a complex symbiotic ecosystem.

> One of the most transformative platforms for innovation is open source. The solution to helping solve problems in the world not just technology problems but social and political problems can and should benefit from open source.
>
> Mark Hinkle, Citrix

Software development increasingly takes place in organizations and communities involving many people. In addition to traditional approaches such as in-house software development (insourcing), there is an increasing trend towards globalization with a focus on collaborations *with* and *within* communities, which may be *known* or *anonymous*. Open Source Software (OSS) has had a dramatic impact on the software industry, albeit initially approached with much scepticism and fear. Today, many organizations adopt OSS in multiple ways and increasingly rely on OSS communities for a steady stream of updates for open source products. Open-source-inspired strategies such as crowdsourcing (Stol & Fitzgerald, 2014) and innersourcing (Stol et al., 2014) are also gaining considerable attention and are becoming viable approaches. Figure 6 summarises the various software sourcing strategies from a customer's perspective. We position these in a *circumplex* based on two dimensions: *control of the product offering* and the *extent to which a workforce is known.*

Quadrant I (Q.I) contains traditional approaches to software sourcing: **insourcing** is in-house software development with a clearly defined workforce, and 'traditional' **outsourcing** involves a workforce initially "unknown" since outsourcing suppliers are often a black-box for customers, but given sufficient time a relationship and trust can develop. In both QI strategies customers have a considerable degree of control.

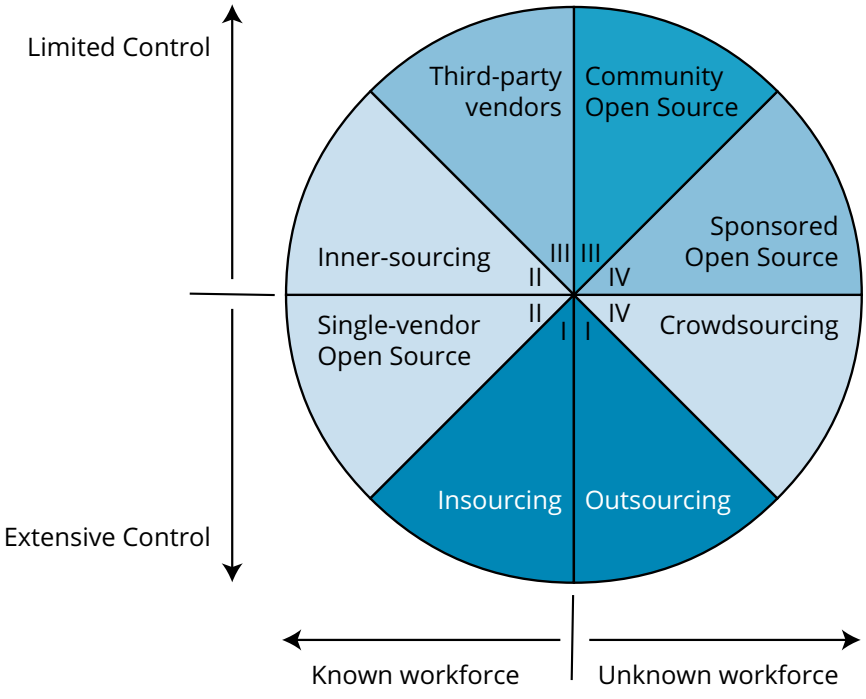

Fig.9: Sourcing Strategies for Software Development (Stol 2016)(Key: Segments in darker shades signify areas which are to date quite well understood in research and practice, while lighter shaded segments represent areas not yet well understood)

Quadrant II (Q.II) contains **single-vendor open source**, where one organization owns and controls an OSS product, as with MySQL and Eclipse, and **inner-sourcing**, where an organization adopts OSS development principles for its internal development. This approach is gaining considerable interest from companies such as **Allstate**, **PayPal**, **Rolls-Royce**, **Samsung** and **Sony Mobile** (Stol et al., 2014). Inner source facilitates ad-hoc collaborations between organizational units that otherwise would not collaborate, creating a culture of transparency and collaboration. Because inner source relies on motivated individuals and self-selection of tasks, an organization has limited control (by design) over the software being developed, management's role is that of empowerment.

Q.III contains **third-party vendors** and **community OSS**. The former happens in software ecosystems whereby independent parties offer extensions or new functionality (*Apps*). Platform providers have limited control over the software developed; excluding offerings to a platform (e.g., through an "app store") is the main way to exert such control. Such vendors are necessarily known as they usually advertise their offerings. **Community OSS** refers to "traditional" open source, that is, OSS projects without any formal participation of firms (or non-profits) that can exert control over what is being developed. The workforce is very much unknown since developers are commonly using pseudonyms and little is known about specific individuals. The Debian Project (a Linux distribution) is one example with a strong emphasis on the *free/libre* philosophy without corporate involvement (Michlmayr et al., 2015).

Sponsored OSS (Q.IV) is similar to the single-vendor open source strategy, with the exception that an organization is merely involved as a co-developing party, and has no exclusive ownership, and therefore has limited control over the project as a whole. An example of this strategy is the Linux kernel—one study suggests that over 80% of all kernel development is done by paid developers (Corbet et al., 2013). **Crowdsourcing** is also inspired by open source (Stol & Fitzgerald, 2014) with an unknown workforce, at least up to the point that any post-delivery payments are made to the "winner" of a crowdsourcing competition—even after payment, a customer will learn very little about a "supplier." In such a case a crowdsourcing organization has a significant level of control in terms of required features in a delivered software. Variants are *bounty-sourcing*, whereby a sponsor offers a bounty to implement or fix a specific feature in an OSS project and *Internal* crowdsourcing

In practice, an organization may face a mix of several strategies to develop software. For example, the OpenStack project (offering software for managing cloud infrastructure), involving several global companies such as EMC, HP and Intel, is a sponsored OSS project (Gonzalez-Barahona et al., 2013); together these companies have a considerable level of (collective) control over the project, similar to a single-vendor OSS project.

Most of the research on collaborative software development tends to focus on collaborations within teams, between teams and among organizations (Mistrik et al., 2010). In each of these scenarios, developers are employed, and are thus known and 'controllable' by their respective organizations. Figure 9 indicates a need to focus on what we call ***alternative workforces***, which vary in much more dramatic ways than the more traditional workforces described above. Some but not all developers may be paid, developers may not be aware of each other (e.g., in a competition-based crowdsourcing setting, but also in open source) and the motivation and goals of developers may vary widely as well.

Much research in the OSS space focuses on initial adoption, but the **sustainability** of these initiatives is less researched. Key questions are: *How can sourcing strategies be sustained if an organization has little influence on external workforces?* And *how can organizations build up sustainable relationships with unknown workforces?*

McKinsey (2013) estimate a potential annual economic value of US$3.2 Trillion to US$5.4 Trillion enabled by Open Data in seven domains (education, transportation, consumer products, electricity, oil and gas, health care and consumer finance), building emission reductions of 3 billion metric tons, and 35 hours per year of commuter time saved.

The **control** dimension raises issues such as: governance approaches; ownership of innovation and IP; mechanisms to exert control such as payments; reputation of an actor in community-based development; conflict control and resolution; leadership and power-shifts. OSS communities may suffer from internal disagreements about the future of a project, and cause "forks" of projects which greatly affect a project's sustainability (Gamalielson & Lundell, 2014) because this may split the community of developers, jeopardizing sustainability. Conflict negotiation has also been studied by Scacchi and colleagues (Elliott & Scacchi, 2003; Jensen & Scacchi, 2005). Organizations that start inner source initiatives also adopt the lack of control and rely on empowerment of an internal workforce to self-select those tasks that they deem most useful. However, it is unclear how this "uncontrollable" model of software development impacts the organization's product strategies (driven by market trends and demands).

The **extent of to which a workforce is known or unknown** raises issues such as: understanding goals of workforces, their motivations, beliefs, expectations, awareness, and norms of the workforce (as a heterogeneous group, i.e., these issues may vary per individual) versus those of a customer seeking to 'source' software; and the ability to retain knowledge and intellectual resources. One example of how these issues can disturb relationships between a 'customer' and 'supplier' is a misalignment of goals or motivation; OSS projects may be started by altruistic individuals, not to offer a fully functional and supported high-quality software solution. Organizations may have different expectations and assumptions. In a crowdsourcing scenario, the fleeting relationship with 'crowd' developers is a major concern from a knowledge management perspective (Stol & Fitzgerald, 2014). Thus, interacting and collaborating with an unknown workforce raises significant challenges for organizations whose aim it is to deliver commercial software products to a market or their clients.

3.5.1 Open Data Services

Open Data is defined as "data that can be freely used, shared and built on by anyone, anywhere, for any purpose" (http://okfn.org/).

In their Digital Agenda (www.ec.europa.eu), European commissioners listed 4 reasons for promoting Open Data initiatives, including potential economic gains from new product and service development (estimated to be 40 billion Euros per year in the EU), addressing societal changes, fostering citizens participation, and improving internal efficiency.

Early research on Open Data (and more generally on Public Sector Information), mainly concerned e-Government inquiries, addressing aspects of democratic theory, voter participation, democratic deliberation, and open government in a broader context. More recent research explored Open Data as a foundation and catalyser of innovation (Lakomaa and Kallberg, 2013), and particularly service innovation. This has led to the introduction and structure of a new research stream named Open Data Services and is giving a structure to the investigation of Open Data as a foundation of service innovation from an Information Systems perspective.

For Open Data to become valuable there needs to be an integrated process that supports the entire data lifecycle management: from raw data collection and characterization with respect to quality, provenance, and legal usability, to its publication and accessibility as information via adequate safe and secure services, to a rich platform of analysis, aggregation, presentation and visualization in ways that make it useful for users to interpret as information.

Scientific Workflows are a direction of research that addresses the infrastructure, the analysis platforms, the interoperability, the curation, monitoring, maintenance, and governance of such (open) Data Services. Initially dominated by ad-hoc scripting and data-flow approaches to combinations, it is now evolving towards more mature platforms that are model driven, allow advanced control structures and fine grained governance models, support semantically enhanced integration across heterogeneous data models, layers and tools, and allow a mature knowledge management and for "in silico" experimentation in the life sciences, social sciences, and healthcare.

Prior work funded by DG CONNECT on services can clearly be used as a foundation for enhancing, exploiting and popularizing Open Data Services. The unique issues of Open Data (availability and access, re-use and redistribution, and universal participation) are likely to pose particular issues for service composition, security, and widespread applicability.

4 Evolving Critical Systems

It is evident that further research is needed that focuses on the development and maintenance of Evolving Critical Systems (ECS). With the increasingly blurred line between hardware and software, at least at design time, where also hardware components exist in form of analysable models, this is an area where Europe can take a lead.

This research must concentrate on the techniques, methodologies and tools needed to design, implement, and maintain critical software systems that evolve *successfully* (without risk of failure or loss of quality). Such research is essential to success and competitive advantage for the EU in IoT, Smart-*, Industry 4.0, and all of the domains and industries identified in Section 1.

In order to understand the challenges of ECS it is important to consider the complementary domains of Evolving Systems and Critical Systems.

Evolving systems[12] may

- have evolved from legacy code and legacy systems;
- result from a combination of existing component-based systems, possibly over significant periods of time;
- be the result of the extension of an existing system to include new functional requirements;
- evolve as the result of a need to improve their quality of service, such as performance, reliability, usability, or other quality requirements;
- evolve as a result of an intentional change to exploit new technologies and techniques, e.g., cloud, service-oriented architectures, or a move towards multi-core-based implementations;
- adapt and evolve at run-time in order to react to changes in the environment or to meet new requirements or constraints, such as regulations or the exploitation of Open Source software initiatives.

Most software systems nowadays are evolving systems, either large and complex, or simple (such as apps) that users expect to add new/modified functionality often, or Open Source, where contributors can make regular additions. The alternative to system evolution is total replacement, often not feasible for cost and other reasons (Hinchey and Coyle, 2009).

Critical systems are systems where failure or malfunction will lead to significant negative consequences (Lyu, 1996). These systems may have strict requirements for security and safety, to protect the user or others (Leveson, 1986). Alternatively, these systems may be critical to the organization's mission, product base, profitability or competitive advantage. For example, an online retailer may be able to tolerate the unavailability of their warehousing system for several hours in a day, since most customers will still receive their orders when promised. However, unavailability of the website and ordering system for several hours may result in the permanent loss of business to a competitor. Our definition of "critical system" includes **safety-critical** and **security-critical** systems, but also **business-critical** and **mission-critical** systems.

12. *Lehman (1980) called these E-type systems.*

4.1 ECS and Software

ECS can be viewed as a special case of the broader software and system design discipline. Similar issues and questions must be addressed within ECS as in other (non-ECS) software research, but with the added (and conflicting) requirements of predictability/ quality and the ability to change.

The IEEE Computer Society's "Software Engineering Body of Knowledge" (SWEBOK) characterises the elements and boundaries of the software engineering discipline (Abran et al., 2004) by defining ten Knowledge Areas (KAs) that are recognised as being core to the discipline. ECS can be considered from a similar perspective:

While ECS is related to each of these Knowledge Areas, a tenth Knowledge Area, *Software Maintenance*, is most obviously relevant. Software Maintenance concerns the changing of a software system – the processes and activities concerned with changing software, as well as specific techniques undertaken during maintenance, including program comprehension, reengineering, and reverse engineering.

4.2 Related Work in Software Evolution

The problem of how to modify software easily without losing quality was widely understood and discussed at the NATO Software Engineering Conference in 1968 (Naur & Randell,1968). Lehman et al.'s early work on the continuing change process of the IBM OS360-370 operating systems and the work that followed from that led to a large body of research into software evolution and the formulation of eight "Laws of Evolution" (Lehman & Belady, 1985, Lehman & Fernández-Ramil 2006). Swanson (1976) identified three types of evolution:

1. *corrective maintenance*, used to overcome processing failure, performance failure, and implementation failure;
2. *adaptive maintenance*, which would overcome change in data environment (e.g.,restructuring of a database) and change in processing environment (new hardware, etc); and
3. *perfective maintenance*, which would improve design, which might overcome processing inefficiency, enhance the performance, and the system's maintainability.

Rajlich & Bennett's (2000) staged-life cycle model highlighted the maturity of a software system as being an essential consideration when planning change. More mature software, where many (or all) of the key developers are no longer in place is seen as being harder to evolve than newer software supported by its original developers.

As software evolves in terms of functionality, it often degrades in terms of reliability. While it is normal to experience failures after deployment and the goal of much of software maintenance is to remove these failures, experience has shown that evolution for new functionality and evolution for maintenance can both result in "spikes" of failure (cf. Figure 4). Over time, a traditional system degrades as it evolves and more, rather than fewer, failures are experienced (Lehman, 1996, Parnas, 1994, Rajlich & Bennett, 2000).

Dynamic evolution (sometimes called run-time or automatic evolution) is a special case whereby certain critical systems may need to change during run-time, e.g., by hot swapping existing components or by integrating newly developed components without first stopping the system (Buckley et al., 2005). This has to be either planned ahead explicitly in the system or else the underlying platform has to provide a means to effectuate software changes dynamically. In terms of the software evolving itself automatically, there are a number of challenges beyond those faced when a human drives the process. Ubiquitous computing systems or autonomic systems are often typified as consisting of large numbers of distributed autonomic, often resource-constrained embedded, systems. These types of systems could be hoped to evolve dynamically but as Baresi et al. (2006) point out, in these domains open world assumptions about how a piece of software might be used are dominant. Designers cannot fully predict how a system behaves and how it will interconnect with a continuously changing environment. Therefore open assumptions must be built in and software must adapt and react to change dynamically, even if such change is unanticipated.

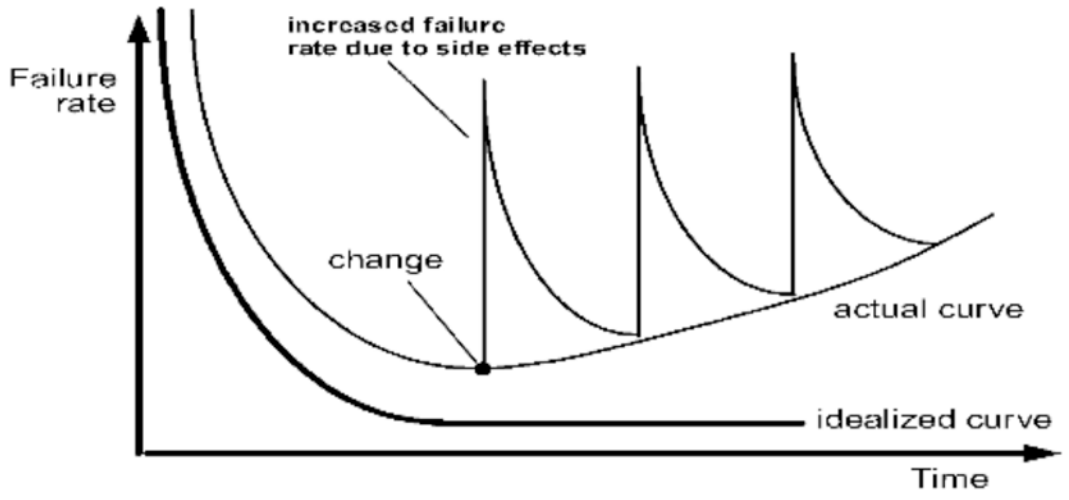

Fig. 10: Wear vs Deterioration (Pressman, 1997).

4.3 Research Questions

We believe that the topic of Evolving Critical Systems is highly relevant. As the ubiquity and complexity of software increase, a requirement has emerged for critical software which can successfully without loss of quality—software that is engineered from the start to be easily changed, extended and reconfigured, while retaining its security, its performance, its reliability and predictability.

The fundamental questions remain:

1. How can we design, implement, and maintain critical software systems that
   A. are highly reliable; and
   B. retain this reliability (or even *improve* their reliability) as they evolve *without* incurring prohibitive costs?
2. How can we maintain critical software quality when its teams, processes, methods, and toolkits, are in a state of constant change?

5 Recommendations for the Commission

There is a widespread perception that although Europe is very strong in terms of contributions to leading edge research and innovation, most of the value from such knowledge is harvested in the US. This is due in large part to a circular and self-fulfilling phenomenon in the more entrepreneurial climate in the US where venture capital, skilled work-force, and digital-friendly customer markets are in strong supply. There is also the undeniable fact that, in Europe as well as globally, companies are taken more seriously if they have a US address, considered the epicentre of technology. This in spite of the fact that the best education in highly competitive areas such as, e.g., formal methods is European, as witnessed by the fact that EU graduates in those disciplines are hired in high numbers by US companies and research institutions, a systematic brain-drain. These recommendations intend to leverage what is unique to the EU, for it to remain an innovative research space and possibly to increase its competitiveness also on the exploitation and uptake.

5.1 Context

We may increasingly see our world as a *Cyber-Physical-Social* system (or a socio-technical system) whereby the Cyber world (namely, software) interacts with physical devices (sensors, actuators, robotics, physical machinery, medical devices, etc.) and where people share much of the data/information retrieved (such as received from sports performance monitoring devices, financial applications, etc.) with their friends and collaborators (and others they do not anticipate) via social media and other collaboration mechanisms.

There is a constant demand for greater functionality, faster performance, and greater ranges of analytics. The demand for more complex, large-scale computer systems is growing exponentially, with many advances in Smart-Cities, Smart-Grids, Transportation, Entertainment, FinTech, Industry 4.0, and many more areas highly reliant on software. Software is also the source of innovation in ubiquitous systems, cloud technology, mobile devices, smart manufacturing, and many more domains of application. There is a significant "push" demand for computing resources, all of which are enabled (and to some extent limited) by software.

It is essential that continued software research helps to raise the practice of software and system development to a fully–fledged professional discipline (similar to the other sciences and engineering) rather than the "craft" (reliant on a limited pool of talented and educated professionals, and a much larger pool of software development "immigrants", with limited training in coding, or even less) that it is today. Without such a rise in qualification profile and competence expectations, innovation will be stifled and many industries currently reliant on software for innovation will fail to meet their potential. The larger, more conceptually demanding, and more connected become the software products, the less they are amenable to the coding first, trial and error-based development approach widely practiced today by decision makers and a programmer workforce that underestimate alike the importance and impact of both high quality and speed. Top quality and increased productivity are particularly essential in "high-tech" Europe, where loss of innovation will mean loss of competitiveness and ultimately economic lost opportunity.

This migration requires that the Commission funds research applying in domains important to initiatives

5.1.1 Commission Support

such as Digital Single Market, Digitising Industry and Open Science, with relevance in cyberphysical systems, FinTech, Industry 4.0, Internet of Things and many other domains. Such research must support:

- Developing new models and paradigms to enable the next generation of software development and higher level languages adequate for both target tiers: **subject matter experts** who participate in the innovation and design, but do not code, and **skilled IT professionals** able to create innovation within the IT systems and their production lifecycle;
- Enabling scalable, tool-supported, and efficient, development methods that address specific domains, industries, organisations and processes (as above);
- Enabling active participation by customers in the software ecosystem and making software development customer-led (need-"pull" rather than technology "push"), while ensuring security and privacy of personal information, particularly in light of Cloud computing technologies, open source software, open data, and social media and other platforms;
- Enabling (via tools, technologies, languages, and paradigms) speedy and cost-effective development of highly-reliable software, able to express *physical*, *cyber* and *social* design objectives simultaneously, but that is also able to evolve without loss of reliability nor prohibitive cost.

5.1.2 International Experience

The US approach to research funding by the NSF is instructive. Open source process research can still be funded in research proposals generally across the program. However, the NSF have declared higher level research objectives in which the open source paradigm is a key facilitator. This is evident in their definition of **Cyberinfrastructure Framework for 21st Century Science and Engineering (CF21)**[13]. The latter recognises the fundamental changes being brought about in all disciplines of science and engineering by cyberinfrastructure. The open-* model is seen as a key component in this, with specific programs to fund software research; for example, **Software Infrastructure for Sustained Innovation (SI²)**[14] and **Data Infrastructure Building Blocks (DIBBs)**[15]

13. *http://www.nsf.gov/pubs/2010/nsf10015/nsf10015.jsp*
14. *http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=503489&org=ACI&from=home*
15. *http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=504776&org=ACI&from=home*

5.2 Recommendation regarding Open-*

5.2.1 Status Quo

There is little point in EU initiatives that seek to mirror US ones – rather, a more holistic big-picture approach is likely to deliver more benefit. This requires that one take the open source software (OSS) success as a given and seek to leverage the open-* paradigm that it underpins and to which it serves as a proof of concept.

EU-funded programmes have resulted in the creation of a very large amount of software. However, much of this software does not survive beyond the life of the project being *funded.* While *releasing such software as open source was seen as a way of ensuring better longevity* and a positive approach, see (Aigrain, 2005), however this licensing per se has little effect without a development community sufficiently socialized to work together in the long term towards common goals. Contrary to the early days, commercial companies now increasingly understand, trust, and embrace both open source software and the business models that allow it to be included in their software stack.

Commercial software providers are now providing a major boost to open source software. However, as too often seen, IPR issues in these proposals are often at odds with open source releaseability. Also, sustainability is linked to the contributor community's creation, organization, and its resourcing, all these rarely available in or after EU projects.

In the big picture, **specific cultural** benefits that arise through the open source paradigm are more important than the software release, for example greater innovation through the variety of broad and deep knowledge within a vibrant open source community, or an accelerated learning curve for new developers by the transparency of open source code.

Higher level strategies that go beyond the open source software phenomenon *per se* and seek instead to leverage what is afforded by the open paradigm are thus more likely to deliver in the longer term.

The proposal cites the ISTAG report "Toward a Strategic Agenda for Software Technologies in Europe" as indicating that such an initiative was needed.

A prior proposal for an Open Source Innovation Platform in Work Programme 2016-2017 aimed to provide a common repository with mechanism to ensure software quality and support re-use, and innovation actions to transform initial open source solutions into commercially viable applications, build communities, and promote reuse of code by new projects.

One of the main recommendations of that report was that "Europe should encourage the emergence of open source software repositories associated with development or qualification tools to gather and foster the result of cooperative R&D or local initiatives".

The proposal argued that OSS emerging from research projects are "rarely properly validated and documented, which means that the scope for re-use is usually lost at the end of the funding period." We argue that this view is somewhat dated and the situation has changed over the years.

The proposal also argues that the solution (an Open Source Innovation Platform) is a vendor-neutral OSS Innovation Platform for European software developers, in particular for OSS developed using H2020 support. We argue, however, that the software market is global and software development is a global activity. As such, creating EU-specific innovations may not be any guarantee of success.

Moreover, there is the issue of how to convince successful OSS project to migrate to the new platform. The effort to migrate between services is significant with no clear benefit. In particular, there is no "killer feature" that would enthuse projects to move. Perhaps offering some additional support (free Cloud CPU and storage, etc.) may help in convincing.

The tools proposed as part of the innovation are not well defined and seem very generic, again giving projects little reason to migrate.

Vendor neutrality might be seen as a benefit, as all other existing platforms are backed by some company. However, there needs to be a compelling advantage for choosing a new infrastructure, when Github has established such a strong presence globally and offers significant advantages over, for example AppHub (www.apphub.eu.com), an EU-funded marketplace.

5.2.2 Open-*

The true power of open source arises in the affordances that the open paradigm leads to, what we label here as the open-* (*open star*) paradigm. At the highest level, we could position **Open Science**. Here we are interested in harnessing the power of open to solve intractable problems. Crowdsourcing and citizen science occur here. However, this should go farther and incorporate the idea of open research results. An interesting initiative here is the US project Open Data Factory[16] which seeks to create metadata standards and infrastructure to aid in the description, discovery, and sharing of large datasets from all kinds of sources, both research projects and commercial and public sector initiatives.

This leads naturally to the **open data** concept. It is now commonplace for municipalities and public institutions to make data open, but until that data is transformed into information which is useful and readily accessible to the average citizen, it provides little value to the public. Software ecosystems such as Google's Android platform, and third-party apps are widespread. In the Smart Cities space, many cities have released a variety of data-sets under an open data model allowing citizens to freely develop civic apps. However, such initiatives have not delivered to the extent expected. Significant initial barriers arose in the lack of appropriate governance models for how data should be made available, how benefits could be realized and what impact and value capture can be achieved. Further problems arise in relation to standardisation of data formats and APIs, application discovery and diffusion, efficient reuse, and a tension between collaboration and competition.

Creativity and motivation are required for this transformation. The EU could foster an environment where people are encouraged to create applications which link open data sets by providing sponsorship for hackathon/maker events. In such an event, curators of open data sets would be brought together with students. The curators would be available to describe their data and answer questions about the interface, while the students would be invited to form teams and build prototype applications, with the most innovative and useful ideas being awarded prize money. PegelAlarm (pegelalarm.de) is an example of an application which links multiple open data sources--information about river flow levels from municipalities across Germany--to provide value in the form of an application which shows and predicts river flows along the length of the river. There are many organizations, such as Mozilla (mozilla.org) which are already organizing events to encourage hands-on involvement in creating applications which would probably be willing to promote and organize events around the open data theme if they were given funding.

Another open-* initiative is **open source hardware** (Pearse, 2012; Thompson, 2008). Open source hardware is an electronic hardware design that is publicly available under an open source license. These documents might include schematics and manufacturing steps. There are various associations that certify the completeness and correctness of the hardware design documents[17].

Two main benefits of open source hardware design are the following:

» Easy customization: The transparency of the design makes the customization of the hardware more straightforward. Many devices for different needs can be developed on top of a single open hardware platform such as Arduino[18].

» Security: Some of the security threats in computer systems are related to the faults in the hardware design. Transparency in design would allow a large community remove the security concerns, similar to open source software.

Reproducing hardware from design documents is getting easier and cheaper for individuals and small companies with the increased popularity of 3D printers and other low cost manufacturing tools. In addition, DIY (Do-It-Yourself) approaches by consumers contributes to the popularity of open source hardware. Some might argue that open source hardware would remove the competitive advantage of a company. Companies who are in this market depend on selling their experience as an inventor or keeping ahead

16. http://www.datafactories.org/
17. http://www.oshwa.org/
18. https://www.arduino.cc/

of the competing copycat products in quality. However, the effect of this business model on profitability needs to be researched further.

Open source design documents help the teaching process and help knowledge sharing by researchers in different domains. The research findings obtained by custom hardware might be hard to reproduce by independent researchers. Research findings linked to open source hardware and software will make reproducing results much more easy.

Some of the most successful open source hardware companies are based in the EU. Research funding in this domain could allow Europe establish a competitive advantage in this area. Investigation of successful applications of open source methodologies in hardware design would help in reaching this goal.

The above analysis has led us to the following recommendations for further research funding by the EU:

» **Open science.** Invest in open science initiatives, such as the foundation of non-profit organizations that can facilitate open access publications; whereas all EU funded research must be published in open access journals, this does not solve the problem of commercial publishers extracting a significant amount of funding from the research ecosystem—this funding would be better spent on research rather than paying for commercial open access licenses for the publications. Furthermore encouraging researchers to share their research data and instrumentation through freely accessible platforms can help to foster collaborations and reduce duplication of effort.

» **Open collaboration.** Whereas most attention is focused on open source as the product, open collaboration is a topic that is only now attracting considerable attention from industry. Trends such as innersourcing and crowdsourcing, whereby companies have limited control over the product offerings and workforces that produce those offerings are transforming the software sourcing landscape. The nature of software development is inherently changing towards an open model – and understanding how to foster this is very important to inform future policy.

» **Open data.** Third parties can build very useful tools using public or government data. Unfortunately, such data is usually stored in different platforms without a common API. Government and public data can be stored in a common repository and a common API. USA open data website https://www.data.gov/ is an example of this approach. Version control and curation of these data sources would need a straightforward but comprehensive contribution and review process and policy. Third party vendors which use such data would help to build innovative products for EU countries.

5.3 Final Recommendations

It is our recommendation to DG CONNECT that is must continue to recognize the **broad relevance of software**, and its **interdisciplinary nature**, in a whole range of industries and domains and its importance to innovation therein throughout the European Union. It is not sufficient to allow each industry sector, nor each member state, to set its own research agenda, as this will result in duplication of effort and inefficiencies, leaving Europe in a weaker position vis-à-vis the rest of the world.

The US also takes this view. While the NSF will prioritize particular research areas with revisions to its programme every few years, it is committed to ensuring support and funding for more fundamental research in a broader sense. Similarly, while there are many topics that are not within DG CONNECT's remit, it should continue to support a wide-ranging software and services research programme, such as its remit allows. However, while keeping this broad range, it would be worth emphasizing the criticality of software and its need to evolve. That is why we recommend a research programme that emphasizes the area of Evolving Critical Systems, supporting the development of software-intensive systems that are reliable and retain their reliability as they evolve, in support of innovation in European industry, fostering growth and leading to social inclusion.

6 References

A. Abran, J. W. Moore, P. Bourque, and R. Dupuis, editors. Guide to the Software Engineering Body of Knowledge (SWEBOK). IEEE Computer Society, 2004.

P. Aigrain, Libre Software Policies at the European Level, in Feller, J, Fitzgerald, B, Hissam, S, and Lakhani, K. (Eds) *Perspectives on Free and Open Source Software*, MIT Press, Cambridge, MA. pp. 447-459, 2005.

L. Baresi, E. D. Nitto, and C. Ghezzi. Towards open-world software: Issue and challenges. In Software Engineering Workshop, 2006. SEW '06. 30th Annual IEEE/NASA, pp. 249–252, April 2006.

J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel. Towards a taxonomy of software change: Research articles. J. Softw. Maint. Evol., 17(5):309–332, 2005. ISSN 1532-060X.

J. Corbet, G. Kroah-Hartman and A. McPherson, *Linux Kernel Development: How Fast It is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It*, **2013**.

M.L. Despa, The Adaptive Nature Of Managing Software Innovation, Journal of Information Systems & Operations Management 7 (1):184-191S.

S. Dobson, A. Denazis, D. Fernández, E. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli. A survey of autonomic communications. ACM Trans. Auton.Adapt. Syst., 1(2):223–259, 2006. ISSN 1556-4665.

B. Fitzgerald, The Transformation of Open Source Software. *MIS Quarterly* 30(3), 2006.

B. Fitzgerald, Software Crisis 2.0, *IEEE Computer*, 45(4), April 2012

J. Gamalielsson and B. Lundell, Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved? J *Sys Soft* 89, 128-145, 2014.

J.M. Gonzalez-Barahona et al, Understanding How Companies Interact with Free Software Communities, *IEEE Software* 30(5), 2013.

D. Grier, Do We Engineer Software in Software Engineering, 2015, https://www.youtube.com/watch?v=PZcUCZhqpus

A. Gurrin, A. Smeaton and A. Doherty, LifeLogging: Personal Big Data, 2014, DOI: 10.1561/1500000033.

M. Hinchey and L. Coyle, Evolving Critical Systems, Lero Technical Report Lero-TR-2009-00, Lero-the Irish Software Engineering Research Centre, 2009.

K. Kirkpatrick, Software-Defined Networking, *Communications of the ACM*, 56(9):16-19, 2013.

E. Lakomaa, J. Kallberg, Open Data as a Foundation for Innovation-The Enabling Effect of Free Public Sector Information for Entrepreneurs, IEEE Access, 1-1, 2013

M. Lehman. Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, 68(9):1060–1076, Sept. 1980. ISSN 0018-9219.

M. M. Lehman. Laws of software evolution revisited. In EWSPT '96: Proceedings of the 5th European Workshop on Software Process Technology, pages 108–124, London, UK, 1996.Springer-Verlag. ISBN 3-540-61771-X.

M. M. Lehman and L. A. Belady, editors. Program evolution: processes of software change. Academic Press Professional, Inc., San Diego, CA, USA, 1985. ISBN 0-12-442440-6.

M. M. Lehman and J. C. Fernández-Ramil. Software Evolution and Feedback: Theory and Practice, chapter Software Evolution. John Wiley & Sons, 2006. ISBN 0470871806.

N. G. Leveson. Software safety: why, what, and how. ACM Comput. Surv., 18(2):125–163, 1986. ISSN 0360-0300.

M. R. Lyu, editor. Handbook of software reliability and system reliability. McGraw-Hill, Inc. Hightstown, NJ, USA, 1996. ISBN 0-07-039400-8.

T. Margaria and B. Steffen. From the How to the What. In "Verified Software: Theories, Tools, Experiments",1st IFIP TC 2/WG 2.3 Conference, VSTTE 2005, Zurich, Switzerland, pp.448-459, LNCS 4171, Springer Verlag.

T. Margaria and B. Steffen. An Enterprise Physics Approach for Evolution Support in Heterogeneous Service-Oriented Landscapes. In 3gERP Workshop 2008, DIKU, Copenhagen, Denmark, November 2008, www.diku.dk/~henglein/3gERP-workshop-2008/papers/margaria-steffen.pdf

T. Margaria and B. Steffen. Simplicity as a Driver for Agile Innovation. Computer, 43(6): 90-92,

June 2010, DOI Bookmark: http://doi.ieeecomputersociety.org/10.1109/MC.2010.177

I. Mistrik, J. Grundy, A. van der Hoek and J. Whitehead (Eds.), *Collaborative Software Engineering*, Springer, New York, pp. 307-328, 2010.

P. Naur and B. Randell, editors. Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO, 1968.

D. L. Parnas. Software aging. In ICSE '94: Proc. 16th international conference on Software engineering, pp.279–287, Los Alamitos, CA (USA), 1994. IEEE Computer Society Press. ISBN 0-8186-5855-X.

J.M. Pearce, Building research equipment with free, open-source hardware. *Science* 337.6100 pp 1303-1304, 2012.

M. Prensky,. *Digital game-based learning*. New York: McGraw-Hill, 2001.

R. S. Pressman. Software Engineering: A Practitioner's Approach, 4 ed. McGraw-Hill, 1997.

V. Rajlich and K. H. Bennett. A staged model for the software life cycle. Computerm 33(7):66–71, 2000.

K. Stol et al., Key Factors for Adopting Inner Source. *ACM TOSEM* 23(2), 2014.

K. Stol, B. Fitzgerald, Two's Company, Three's a Crowd: A Case Study of Crowdsourcing Software Development. Proc. *ICSE'14, Hyderabad, India*, 2014.

K. Stol and B. Fitzgerald, Why and How Open Source Projects should adopt Time-Based Releases. *IEEE Software* 32(2), 2015.

K. Stol, Managing Software Sourcing with Alternative Workforces: A Holistic Overview and Research Agenda, Lero Technical Report, 2016.

E. B. Swanson. The dimensions of maintenance. In ICSE '76: Proceedings of the 2nd International Conference on Software Engineering, pages 492–497, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

J. Teixeira, Understanding Competition in the Open Source Arenda: The Cases of WebKit and OpenStack. In Proceedings of OpenSym'14, Berlin, Germany, 2014.

C. Thompson, Build it. Share it. Profit. Can open source hardware work?. *Wired,* 2008.

P.M. Valkenburg and J. Peter, Adolescents' identity experiments on the Internet: Consequences for social competence and self concept unity. *Comm. Res*. 35(2): 208-231, 2008.

D. Vodanovich, D. Sundaram, and M. Myers, Digital natives and ubiquitous information systems, *Information Systems Research*, 21 (4), pp. 711-723, 2010.

J. Yan and X. Wang, From Open Source to Commercial Software Development. In Proceedings of International Conference on Information Systems, *ICIS 2013.*

Chair of Service and Software Engineering, Universtät Potsdam, Germany, margaria@cs.uni-potsdam.de

Chair of Programming Systems, TU Dortmund, Germany, steffen@cs.tu-dortmund.de M. Michlmayr, B. Fitzgerald and K.

Lero THE IRISH SOFTWARE RESEARCH CENTRE

European Commission