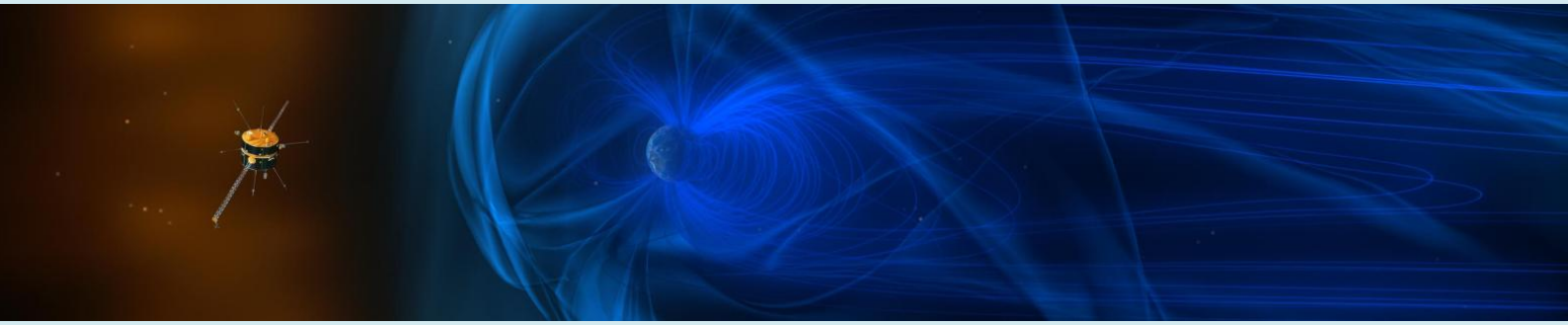
Research Priorities in the area of Software Technologies



Prepared by Diomidis Spinellis

European Commission

A report prepared for the EU DG Communications Networks, Content and Technology — E2
PO 30-CE-0751856/00-91

Diomidis Spinellis
Professor, Department of Management Science and Technology
Athens University of Economics and Business
http://www.dmst.aueb.gr/dds

Version 5.3 — November 2016

CONTENTS

EXECUTIVE SUMMARY

In the words of the web's inventor Marc Andreessen, "software is eating the world". Ever more products, services, and entire industries, existing ones as well as new, are running on software. This report, based on studies surveying the evolution of technology as well as journal articles, conference papers, and talks covering the future of software engineering, argues that significant investment in software engineering research can help Europe stay on top and even lead a world that is increasingly defined and shaped by software.

The need for targeted research in software engineering is prompted by developments in three broad areas. First, **the computing landscape is changing** from top to bottom. Cloud computing and infrastructure as code allows the radical rethinking of computing systems; universal memory obsoletes current memory hierarchies; multicore architectures change how energy efficiency and performance are to be obtained; while big data, machine learning, and natural user interfaces open new possibilities for computing applications. Software lies at the heart of all these changes. Second, seven **software-driven vertical application domains are reshaping entire industries** and society as a whole. These domains are autonomous vehicles, massive open online courses, open intellectual property, the Internet of Things, life sciences, 3D printing, financial technology, and Industry 4.0. Third, the computing landscape's technological trends and the changes in the vertical application domains, give rise to **several critical crosscutting software engineering challenges**. These challenges involve the taming of the immense scale and complexity of the required software through suitable tools and abstractions, data analytics, novel processes and collaboration mechanisms, as well as the integration of software development with operations. On top of these, return with a vengeance known challenges regarding security, privacy, reliability, and environmental sustainability.

Software engineering research in the areas of **software construction**, **software design**, and **software engineering process** must be funded as a specific priority, so that it can act as a foundation for the robust evolution of computing and its applications. The findings of such research, whether empirical, such as the effect of code size, testing, or reviews on software quality, or technological, such as static program analysis tools and model checkers, increase the IT industry's efficiency and benefit society through more and higher quality software. In contrast, restricting software engineering research to the incidental support that takes place within the development of applications, is like trying to build a national highway network by patching together cobblestone paths of small villages. Such an approach is ineffective and drags down the economy.

Countries around the globe adopt diverse approaches to lead in the area of software engineering. In Europe, Germany and the UK target funding specifically to software engineering research. In the US the focus is toward building scientific and engineering software infrastructures, in China research is centered on data-driven software technologies,

while in India private funding for software engineering research aims to serve the requirements of a large software development industry.

Europe can build on its world-class pockets of excellence in specific areas of economic activity, such as automotive manufacturing, engineering, aerospace technology, financial services, and luxury goods marketing. Thus, focused, significant, and effective research funding in the area of software engineering can help the development of new methods, tools, architectures, systems, business models, processes, and applications that can be instrumental in the establishment of Europe as a center for the rise of a software-run economy.

1 INTRODUCTION

In the words of the web's inventor Marc Andreessen, "software is eating the world". Ever more products, services, and entire industries, existing ones as well as new, are running on software. This report, based on studies surveying the evolution of technology as well as journal articles, conference papers, and talks covering the future of software engineering, argues that significant investment in software engineering research can help Europe stay on top and even lead a world that is increasingly defined and shaped by software.

The vision proposed in this report is that targeted software engineering research can act both as an *enabler* for the flourishing of other rapidly evolving technologies and application areas and as a *catalyst* for these developments to take place. This approach is required for a number of reasons.

- Failing to adjust software products and development methods to new technological developments will rapidly lead to obsolescence. Past examples include market leaders such as IBM in computers as well as Motorola and Nokia in mobile phones, which have lost their dominant market position. In contrast, proactively funding software research in the areas of new developments can establish new business and growth opportunities.
- Emerging vertical application domains are going to disrupt entire industries. Most of them are driven by software. However, existing market players often fail to appreciate software's critical role while new entrants typically lack the resources to invest in the required software engineering research. Targeted software research funding can provide solid foundations for building the applications in the new vertical domains at the standard required to establish market leadership.
- Finally, failing to address the cross-cutting concerns of rising scale and complexity, tool support, and security discussed Section 4, can throw sand in the workings of any new developments, halting advancement through increasing costs and failures as well as lowering productivity and public trust.

2 METHOD

Input for this report came from dedicated studies, journal articles, conference papers, and talks covering the future on software engineering. All are listed in the *Bibliography and References* section (0). In summary the most important sources were the following.

- IEEE Computer Society 2022 Report [AFF+14, AFF+15].
- EU's "Knowledge Future report [HBD+14].
- The World Economic Forum "Deep Shift" survey on how software is changing the world [Esp15].
- Papers from the "Future of Software Engineering" Track at the *36th International Conference on Software Engineering* [Her14].
- Articles from the *IEEE Software* theme issue on the "Future of Software Engineering" [SCC+16].
- Grady Booch's keynote address on the "Future of Software Engineering" delivered at the *37th International Conference on Software Engineering* (2015).
- Finally, the voice of the practitioners regarding future development was taken into account through qualitative and quantitative data reported in a published paper [BZ14].

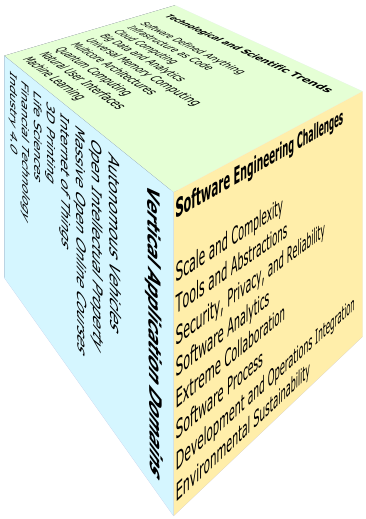


Figure 1: Drivers of software engineering research

From these sources it becomes evident that software engineering research is required to address three types of developments, as illustrated in Figure 1: scientific and technological trends, such as multicore architectures and universal memory computing— see Section 0; emerging vertical application domains, such as autonomous vehicles and health science — see Section 0; and, most importantly, cross-cutting software engineering challenges that arise from the other two areas, such as rising scale and complexity, tool support, and security — see Section 0.

TECHNOLOGICAL AND SCIENTIFIC TRENDS: INFLUENCE ON SOFTWARE DEVELOPMENT

Technology influences the way we develop software both through the affordances it provides (for example ample directly addressable main memory and multiple computing cores) and through the requirements it imposes (for instance the ability to analyze mountains of data). Second order effects also come to play when technological trends converge and feed on each other. One such example includes the convergence of cloud computing, big data analytics, and future networks. The following paragraphs outline the main technological trends that will affect software development in the next five to seven years and the corresponding challenges associated with software engineering.

SOFTWARE DEFINED ANYTHING AND INFRASTRUCTURE AS CODE (SDX/IAC)

Advances in raw hardware power and virtualization technologies are increasingly allowing the definition of sophisticated processing and storage nodes through software. Particular instances of this phenomenon include *software-defined networking* (SDN), *software-defined storage* (SDS), and, on a larger-scale, *software-defined data centers* (SDDC). The common idea behind this *software defined anything* (SDx) trend is the provision of relatively dumb, but powerful and affordable hardware, and the implementation of higher-level functionality through **configurable software**. As a particular use case, consider software-defined networking, where the network's control plane is implemented through software. A network switch is a simple and efficient forwarding table, and all additional functionality, such as routing and access control, is handled through (potentially centrally managed) software components. Other devices that are mainly defined through software include televisions, cameras, phones, engine and vehicle management systems, smart electricity grids, and home automation nodes.

From a software engineering perspective, the phenomenon is sometimes described through the term *infrastructure as code* (IaC). This entails large and small-scale infrastructure whose design and implementation is expressed through means that are indistinguishable from software code. As an example, consider a data center setup expressed in a system configuration management language, such as Puppet, Chef, or Ansible. The opportunity associated with this phenomenon is that such code must be developed and managed through processes that are essentially part of software engineering. The corresponding challenge is that the underlying artifacts do not behave as traditional code: for instance, debugging a software-defined data center is very different from debugging a Java program. Furthermore, developers in this area are likely to be trained as domain experts rather than software engineers. Instilling a software engineering culture will be an uphill but rewarding struggle. Consequently, the corresponding research challenges are the following.

- Establish software engineering processes for dealing with infrastructure as code.

- Define techniques for evaluating and controlling the quality of IaC artifacts.
- Create tools, such as static and dynamic analyzers, debuggers, testing frameworks, package managers, and IDEs for developing systems based on IaC.

CLOUD COMPUTING

Cloud computing refers to the abstracted and on-demand provision and utilization of computing resources. It encompasses the provision of

- unconfigured (virtual or real) hardware (processing nodes, storage, routers, load balancers) under the name *infrastructure as a service* (IaaS),
- middleware (runtime, application and web servers, databases, queues, directories) under the name *platform as a service* (PaaS), and
- complete applications (collaboration, office productivity, CRM, HRM, accounting) under the name of *software as a service* (SaaS).

Although significant progress has been achieved in this area over the past decade, the extent of applications that are not benefiting from this technology indicates that the uptake's scale will result in a **qualitative** rather than simply quantitative change in the field of software engineering.

A major task that software engineers must address, is the **friction-free integration** of IaaS, PaaS, and SaaS offerings with minimal vendor lock-in. This will benefit from cloud computing standardization and interoperability efforts, which will in turn require the definition of high-level services, APIs, and other abstractions. The corresponding research challenge is to devise suitable models, architectures, and design patterns that can address these needs. A challenge closer to the end-users is the ability to easily create reliable and efficient cloud computing mash ups based on software services and data.

The amount of innovation and growth that is likely to happen in the area of cloud computing, will spur the creation of a **cloud-based economy** where services, data, computing capacity, bandwidth, and other inputs will be traded, often in real time, based on current and projected demand and supply. Software engineers will be required both to define and implement the building blocks for the cloud-based economy, and write software that will efficiently utilize the provided resources. This will require the development of suitable methods, tools, as well as infrastructure, such as middleware, to efficiently utilize the provided resources. Specific research challenges in this area include the following.

- Efficient abstractions and container mechanisms that allow the efficient, scalable, robust, and secure utilization of cloud resources
- Tools for deploying, controlling and monitoring cloud-based services at a planetary scale
- Abstractions, libraries, and middleware for scaling applications deployed on the cloud and for reaping the benefits from economies of scale and scope

Finally, software engineering will need to address the requirements of the part of **high performance computing** (HPC) that will migrate to cloud-based offerings. These are mainly, high granularity scheduling, efficient low-noise virtualization and multitenancy, powerful interconnects, and user-friendly methods to utilize general purpose graphics processing units (GPGPUs). The requirements can be addressed through suitable architectures, libraries, and tools. Other technology changes likely to disrupt HPC include low-power components, such as ARM CPUs, non-volatile memory, and big data analytics. All of them benefit from software engineering in ways that are detailed in other sections of this study.

BIG DATA AND ANALYTICS

Personal mobile computing devices, digital social networks, scientific research, e-commerce, and the internet of things are generating a vast amount of data. Storing, processing, and analyzing these data is challenging, but can also be extremely rewarding to individuals, society, and the economy. This is, for example, the impetus behind the European Open Science Cloud, an initiative aiming to provide free, open, and seamless services for the storage, management, analysis, and re-use of research data. Software that handles big data will need to advance in order to cope with the demands of rising size, processing speed, and cost efficiency. This will require the development and deployment of appropriate architectures, frameworks, systems, tools, and libraries.

A specific software engineering challenge in this area include the development of architectures and formalisms that can efficiently express and implement big data processing operations. Another, equally important, research challenge is the introduction of software engineering methods and practices in the processing of big data — big data engineering.

UNIVERSAL MEMORY COMPUTING

The dynamic random access memory (DRAM) technology, which currently serves the vast bulk of computing's main memory needs is reaching the limits of its 40 year evolution. Large trench aspect ratios and small apertures of individual DRAM cells are leading to seemingly insurmountable quality problems. However, new types of memory technologies, such as spin-transfer torque RAM (STT-RAM), phase change memory (PCM), and the Memristor, can offer increased capacities through crossbar layouts and vertical stacking. In contrast to DRAM, these technologies have also the property of being non-volatile memory (NVM). On the secondary storage side, a related technological change is happening through the adoption of (mainly NAND flash memory based) solid state drives (SSDs) in place of magnetic disks.

Both technological changes lead to a type of universal memory that combines the fast random access characteristics of DRAM with the non-volatility of disk storage. This calls for a different programming model that focuses entirely on **main memory computation**. The corresponding software-related challenges are two. The first challenge involves the creation of suitable abstractions and systems (such as operating systems, middleware, database

management systems, libraries, and frameworks) to support this model and benefit from it. In particular, the paradigm shift of main memory computation creates space for new market entrance to establish foothold in the corresponding markets. The second challenge is the development of processes to adapt existing application software to the new technological landscape. Modern organizations depend on a colossal amount of what will become legacy software. Adapting it for the new paradigm will cost dearly, while staying behind leaves the field open to new nimbler players. Architectures, libraries, and tools that can lessen the conversion pain (e.g. by converting a legacy three-tier software architecture to use a main-memory embedded database) can reduce the conversion's cost and provide drastically more efficient applications.

MULTICORE ARCHITECTURES

The main driver for the rise of multicore architectures is the energy cost of higher CPU frequencies. The dynamic switching power consumed by a CMOS microchip is proportional to the cube of its clock frequency. For example, halving the CPU's frequency lowers its dynamic power requirements to 12.5% of the original; doubling the number of cores to compensate for the frequency reduction, will still require only 25% of the original power. In this case, further savings of power can be achieved through the reduction of static power by switching off one of the two cores when it's not required. This trend affects enterprise and high-performance computing occurring in cloud-based servers, which cannot realistically rely on the rise of CPU frequencies to deliver the required computing capacity. It also affects energy-constrained embedded devices, such as mobile phones and IoT nodes, which must be very power efficient.

Currently, a serious obstacle in the use of multicore architectures is the gap between the skills of existing programmers and the skills required to effectively program them. In the words of David Patterson [Pat10]: "parallelism can work when you can afford to assemble a crack team of Ph.D.-level programmers to tackle a problem with many different tasks that depend very little on one another". Thus, on the software side, the rise of multicore architectures drives demand for software and systems architectures that can harness the multiple cores. This includes compiler technologies, parallelization methods, profiling tools, and software components that can efficiently exploit non-homogeneous multicore architectures, such as those with GPUs.

QUANTUM COMPUTING

Quantum computing (QC) is based on the counterintuitive properties of quantum mechanical phenomena. Many algorithms that have formidable cost in classical computing infrastructures can be, in theory, efficiently executed on QC hardware. Small scale experiments have shown that this could be a realistic possibility. Current hardware is still at its infancy, with even the most advanced hardware supporting computation with around one thousand qubits (quantum bits). Yet, both commercial companies and researchers frequently report significant practical and theoretical progress, so this is an area where game-changing developments cannot be ruled out.

Practical QC will significantly affect software development, requiring a complete rethink of how software is expressed and structured (programming languages and architectures), what tools are used for its development (compilers and IDEs), and what software components and abstractions should be provided. Given the difficulty of constructing pure QC computing hardware, another significant area of development is the integration of classical computing with its quantum sibling.

NATURAL USER INTERFACES

The increased availability of multitouch sensors, microphones, accelerometers, and high-resolution cameras on computing devices allows for a radically **different way of communicating** with them. This mode of interaction replaces or supplements the keyboard and mouse with touch, gestures, and speech giving rise to the so-called natural user interfaces (NUIs). Deployed on emerging settings ranging from mobile devices, to cars, to living rooms, these are increasingly used in a real-time networked fashion, providing the NUI makers with rich data regarding the interactions. The challenge for software adopting NUIs is to become increasingly intelligent and transparently responsive. Given the technology's sophistication this capability should be provided at the systems level rather than separately for each application. Specifically, such software should integrate input from multiple sensors, predict user input, adjust according to context, infer intentions, resolve ambiguities through dialogue, and augment the users' perception of reality. The related software engineering challenge concerns the development of reusable components and frameworks that can make this vision a reality.

MACHINE LEARNING

Machine learning (ML) systems learn from data and then operate based on that knowledge. Notable applications include pattern recognition, natural language processing, and recommendation systems. Emerging approaches, such as deep learning, have led to remarkable practical successes in a number of areas. Two software-related challenges need to be addressed to increase the benefits that can be reaped from ML systems. First, the scalability of ML systems must be addressed in order to tame the processor and memory demands of current algorithms. This also entails the harnessing of new upcoming hardware capabilities to support ML computations at the software systems level. Second, to expand the reach of ML systems beyond large vertical areas into a wider range of applications, the ML systems' configuration must be brought closer to the capabilities of non-experts. Research in this area can look at domain-specific languages, specialized IDEs, and agile software development methods tailored to the development of ML systems.

THE EFFECT OF SOFTWARE IN VERTICAL APPLICATION DOMAINS

Software is increasingly becoming a cross-cutting, enabling technology for most information and communication technology (ICT) developments. Software technologies are driving and shaping many vertical application domains ranging from emerging ones, such as autonomous vehicles to long-established ones, such as life sciences. Therefore, research in software technologies is an enabler and a driver for these application domains.

AUTONOMOUS VEHICLES

Autonomous vehicles, once a staple transportation mode only in science fiction movies, are increasingly seen as an upcoming development that will change many aspects of the economy and modern life. An autonomous vehicle is a crucible for most areas of computing and software engineering: from artificial intelligence and signal processing, to sensor networks and user experience. Major challenges associated with software include the harnessing of the system's complexity while achieving the required levels of performance, safety, and reliability.

OPEN INTELLECTUAL PROPERTY

The open intellectual property (IP) movement encompasses information, software, scientific publishing, media, protocols, systems, designs of physical artifacts, online courses, and printable 3D models. Showcase examples include open source software, open access publishing, massive open online courses (MOOCs), and creative commons licensing. The rise of the internet is a showcase example built on many openly available elements, such as the open TCP/IP and the world-wide web protocols, the Linux operating system, the Wikipedia knowledge base, and media uploaded on social networks. The movement is obviously software driven, but its key relationship to software engineering is more subtle. First, it concerns the development of software that integrates open IP and innovation models based on it. Second, it concerns the treatment of many open IP artifacts (e.g. 3D models, MOOC curricula, or legislation) as software in order to benefit from existing and new software development practices [Spi15]. These include collaboration models (e.g. based on configuration management systems and open source software processes), modularization, and the development of reusable components (e.g. Wikipedia's templates). In short, the challenge here is to broaden software engineering theory and practices into the domain-specific engineering of all digital artifacts.

MASSIVE OPEN ONLINE COURSES

Massive open online courses, offer free education to all through the internet. The underlying software of MOOCs is not comparable in complexity to that of, say, autonomous vehicles. However, both developments are equally capable to transform society. Software can have a major role here through the development of authoring platforms and delivery systems that integrate the development of content authoring with assessment, personalization,

specialization, and localization. Furthermore, given the rising opportunities and needs for software development, MOOCs can act as enablers for increasing the number of domain experts and end-users that participate in it. Thus the research challenge here is the use of MOOCs to democratize software engineering.

INTERNET OF THINGS

The Internet of Things (IoT) is driven by the ubiquity and connectivity of sensors and actuators, either as bespoke devices, such as a temperature sensor, or as complete intelligent systems, such as a smartphone. The IoT can provide intelligence to cities, homes, factories, supply chains, and retail shops and thus transform business and everyday life. Two key software related challenges associated with the IoT are the development of architectures, reusable components, and applications that work with peer to peer and intermittent networking connectivity and the development of systems that can harness and exploit the massive amount of data that the IoT generates.

3D PRINTING

3D printing, also known as additive manufacturing, revolutionizes manufacturing, both at the factory floor through new processes and manufacturable artifacts, and, at the level of businesses and individuals, by increasing the availability and reducing the cost of bespoke parts. 3D printing can profit from improved design, solid modeling, and analysis software frameworks and tools. Furthermore, in common with open IP, the 3D printing ecosystem can benefit by treating 3D models as software thus reusing experience, processes, and tools associated with software engineering. This includes the development frameworks, platforms, and reuse models.

LIFE SCIENCES

The domain of life sciences (LS) is extremely broad. It includes large industry sectors, such as pharmaceuticals, health care services, biotechnology, medical technology. Life sciences affect our lives both directly, through the control of human ailments and diseases and indirectly through improved crops and animal health. A lot of progress in the LS field is software-driven, either through the development of algorithms and tools, as in computational biology and bioinformatics or through complete information systems, as in the provision of personalized health care. Software development challenges associated with LS include the management of the field’s gargantuan demand for software, enabling the level of interoperability needed for delivering on many of the field’s promises, handling the massive amounts of generated data, while achieving the required safety, reliability, and protection of personal data.

FINANCIAL TECHNOLOGY

A number of disruptive innovations and services are changing how financial services are structured, provisioned, and consumed [BLBM15]. Payments, which are increasingly becoming cashless, are on the verge of being transformed through new methods, namely mobile payments and those based on crypto-currency (e.g. the bitcoin protocol and block chain databases). Novel platforms and methods for capital raising (e.g. Kickstarter and IndieGoGo) and peer-to-peer lending lower barriers to entry both for businesses and for investors. In Europe, the revised Payment Services Directive (PSD2) can provide new business opportunities. Traditional capital markets are also changing through the application of big data analytics, machine learning, and new exchange platforms. Insurance is another area facing technology-driven disruption. The risks of insurance are better understood through IoT sensors; for example measuring a car's speed and acceleration patterns can provide insights on the probability of accidents. Furthermore, online platforms and the sharing economy allow the disaggregation of insurance services; a person's friends and relatives might have first-hand knowledge of the risks associated with insuring that person, but not the capital required to underwrite them. Block chain databases can also become the basis for the development of decentralized applications in domains other than finance. Finally, investment management is affected on the consumer side from social networks and automation and on the provider side through cloud computing. Dependable and robust software lies at the heart of all these changes.

INDUSTRY 4.0

Close integration between the physical world and its IT models, ubiquitous powerful smart sensors and actuators, and distributed controlling software coordinating through a service-based model are changing the way factories (and the associated businesses) are organized and operate. The creation of value chains from combining cyber-physical systems, the internet of things, and the internet of services has been termed Industry 4.0. Under it products are still manufactured, but a significant part of their value proposition comes from the provision of a service and through mass customization. Products that fail to follow this trend risk becoming a low-margin commodity, as is currently happening with the contract manufacturing of consumer electronic devices.

Software plays a fundamental part in Industry 4.0; the changes associated with it are very much software driven. However, given the industry's existing rigid and risk-averse setup and stringent safety and reliability requirements the challenges facing the software development are formidable. These include the need to set up wide, agile, and open application development ecosystems while still satisfying the products' safety and security requirements as well as the need to rapidly develop and evolve software for the app economy. Research to satisfy these needs involves the development of tools and methods that combine agility with reliability and that enable the transformation of physical based processes into virtual ones expressed by software.

CROSSCUTTING SOFTWARE ENGINEERING CHALLENGES

The technological developments reported in the preceding two sections, together with the convergence of cloud, big data, and software-defined infrastructures (e.g. future networks) create a number of crosscutting challenges in software development. These include the scale and complexity of the underlying computing problems, the necessary tools and abstractions, security and privacy requirements, the development process, and the sustainability of the computing ecosystem.

SCALE AND COMPLEXITY

Computing systems in the coming years will be considerably larger and more complex than existing ones. In contrast to growth as we have seen it up to now, the rise in scale and complexity will not be mainly driven by the isolated growth of individual systems, but from network effects associated with the combination of diverse data sources and computing devices. Systems of systems will become the norm.

Consequently, software development has to harness the friction associated with the rising complexity, perhaps by coming up with methods and systems that interoperate in a more fluid manner than existing rigid APIs. This includes building software with components that may turn out to be unreliable and untrustworthy. In the past we have conquered complexity through reusable software modules. In the future we need similar approaches that can conquer the size of data and the complexity of large system configurations, through the establishment of data and system configuration ecosystems and markets. Finally, agile software development, which has delivered significant insights and improvements within some organizations, needs to be scaled up to work across organizational boundaries. This will require rethinking of existing contracting and cooperation practices.

TOOLS AND ABSTRACTIONS

Progress in tools and abstractions has driven the software industry from its infancy to its current state. Things we now take for granted, such as compilers and object-oriented programming, were once only fledging research results. The new technology-driven challenges must be addressed through the development of suitable programming languages, libraries, frameworks, and architectures as well as testing, debugging, profiling, analysis, and tuning tools. Thankfully, rises in computing and networking capacity allow the realization of approaches that were in the past unrealistically expensive, such as reverse debuggers, microservice architectures, and the individual hardware protection of memory objects. Harnessing the available hardware resources, building on established software ecosystems, and utilizing advances in theory will continue to be major drivers of progress.

With computing touching every aspect of the daily life, relying on professional developers to deliver the required functionality is unlikely to be enough. This shortage can be addressed

through investments in targeted education (e.g. programming bootcamps and MOOCs) as well as through the development of systems, frameworks, APIs, methods, and tools that allow the implementation of applications by domain experts and end users as well as the creation of powerful mash-apps from existing components.

SECURITY, PRIVACY, AND RELIABILITY

Many emerging technologies, such as the IoT, and vertical application domains, such as autonomous vehicles, Industry 4.0, and life sciences, strain to their limit our existing approaches for delivering software and systems that are secure, reliable, and respect individual privacy concerns. These areas are orthogonal to each other and recent research suggests that the underlying phenomena are also independent. However, all three share the trait that they effect software and systems (often dramatically) mainly through their absence, rather than their presence. They are also often at odds with each other as well as with other required software attributes, such as usability, functionality, stability, and maintainability.

All the attributes can benefit immensely from the adoption of appropriate architectures, languages, designs, and tools. In particular, the field's leaders have realized that security and privacy cannot be bolted on after the fact, but must be an integral part of the system's design [ACFD+14]. Therefore, software development research should focus on developing architectures and designs that can inherently deliver the required security and privacy attributes. Sadly, the underlying idea, namely security by design, is more honoured in the breach than the observance. This problem can be addressed through systems-level research into the factors required to adopt the corresponding designs. Progress can also be achieved through the development of tools, such as static and dynamic analysis checkers, libraries, frameworks, languages, and systems that package the functionality in a reusable way, as well as software development processes that demonstrably deliver on those fronts.

SOFTWARE ANALYTICS

It would be very strange indeed if big data and business analytics did not affect software development. The development and running of software and the operation of computing systems generate (or can be made to generate) immense quantities of data. These concern runtime problems (such as application crashes), test results, performance figures, development process metrics, usage profiles, and subjective experience reports (application reviews). Software engineering can (and should) profit both by developing methods for generating and collecting the appropriate data, and by analyzing the data. Results from this endeavour can improve reliability, usability, development efficiency, and profitability.

A lot of work has already been done in this area, but the field is still at its infancy. Important advances are likely to come from the following: the combination of quantitative with qualitative studies; the focus on actionable results that developers can readily use; the combination of static, dynamic , product, and process data; the running of (paid)

experiments in realistic production settings; the development of theory based on the analysis; as well as replication studies.

EXTREME COLLABORATION

The internet has revolutionized many parts of software development. Software package repositories or *forges* act as central points for gathering and locating reusable software components, and allow the development of sophisticated package managers. These package managers in turn, automatically manage cross-package dependencies, deployment, and updates, thus reducing considerably the burden of software reuse [Spi07]. Most of the underlying components are developed by tens of volunteers as open source software. Consequently, many modern software systems easily contain code written by thousands, giving rise to the phenomenon of extreme collaboration. In a commercial setting this mode of work can be further extended by purposely crowdsourcing software and content, though several challenges regarding the resulting software quality need to be addressed. Therefore, further research is needed in the handling of cross-platform dependencies and in the management of the required quality attributes in software built from diverse components.

SOFTWARE PROCESS

Advances in the software engineering process, from the tight management of each development phase's deliverables in the 1970s and 1980s to modern agile development approaches, have been agents of progress and change. Further challenges and opportunities lie ahead. These include the continuous delivery and adjustment of software artifacts in the face of changing requirements, software components, telemetry data, user experience intelligence. Another opportunity comes from articulating, managing and adjusting processes to varying risks and rewards. Over the recent years testing has been recognized as an integral element of software development rather than as a separate activity; increasing its effectiveness remains a challenge. Finally, the emergence of software analytics allows us to conduct empirical studies regarding processes, tools, and quality in use that deliver concrete, actionable results with measurable benefits.

DEVELOPMENT AND OPERATIONS INTEGRATION

The movement of many software services to cloud infrastructures, has helped us realize that developing software and managing IT infrastructures share commonality and offer important collaboration opportunities. For example, IT professionals use build-like tools to automate software deployment and infrastructure updates. This has given rise to the DevOps movement. Compared to its software engineering sibling, the operations part is still in its infancy in many organizations. Therefore, considerable research is required to bring the corresponding processes, languages, and tools to the level of maturity that we now take for granted in software development. Further research is also likely to deliver considerable benefits in the area of collaboration and sharing of assets among teams.

ENVIRONMENTAL SUSTAINABILITY

The final horizontal challenge associated with software development research is that of environmental sustainability: the impact of the systems we build on the environment. This includes the energy efficiency of computing and communications systems, the management of demand and supply for computing at sustainable levels, and a drive toward improved aggregate end-to-end sustainability from components to data centers. Importantly, energy consumption is heavily affected by the software executing on hardware devices. Therefore, research is required into systems that can profile, monitor, and manage energy efficiency from the level of code components to the level of complete data centers.

INTERNATIONAL OUTLOOK

This section contains an overview of software engineering research in three sample countries: USA, China, and India. Obviously research in other countries, such as Japan, Brazil, Korea and individual European countries is equally important, but beyond what could be accomplished within the scope of this study. In sum, the focus in the US is toward building scientific and engineering software infrastructures, in China data-driven software technologies, while in India the requirements of a huge software development industry.

NATIONAL FUNDING IN EUROPE

Various European countries specifically fund software engineering research. Two notable cases are given below.

In **Germany** the national research funding body (Deutsche Forschungsgemeinschaft — DFG) is funding software engineering research through a number of priority programs.[1] These are established by the DFG Senate when the coordinated support given to the area proposed by researchers promises to produce particular scientific gains. The most prominent of these areas are: "Reliably Secure Software Systems" (Zuverlässig sichere Softwaresysteme — SPP1496), "Design for Future — Managed Software Evolution" (SPP1593), "Software for Exascale Computing" (SPP1648), and "Design and Architectures of Dependable Embedded Systems" (SPP 1500). The topics of these areas match challenges and opportunities identified in this report.

In the **United Kingdom** the Engineering and Physical Sciences Research Council (EPSRC) gives as the goal of software engineering research the "development of dependable, efficient and maintainable software" and lists as the area's topics "requirements engineering, software design, software quality (including reliability, safety, security and usability), software testing and analysis, software adaptation and evolution, software process and automation, and empirical software engineering".[2] The research aims to address challenges associated with the shift to mobile devices, cloud computing, distributed systems, and parallel computer architectures. EPSRC finds that software engineering research conducted in the UK is of high quality, but the current funding is insufficient to meet the outlined challenges. Consequently, it is planning to **grow** this area in comparison to others.

USA

Civilian research in the USA focusing on software engineering technologies is described in the National Science Foundation's (NSF) vision of a Cyberinfrastructure Framework for 21st

[1] http://www.dfg.de/en/research_funding/programmes/list/index.jsp?id=SPP

[2] https://www.epsrc.ac.uk/research/ourportfolio/researchareas/software/

Century Science and Engineering (CIF21).[3] Under this vision software is seen as a priority for driving innovation in science and engineering through computation, experiments and theory. The software vision's goals comprise basic software engineering capabilities through a powerful software ecosystem, foundational research, the application of software on scientific research, software development education, and the development of policies to transform current practice. The collective aim is to transform specific research and education innovations into integral cyberinfrastructure software resources.

Research investment is taking place at three scale levels: **software elements** (think the development of a dynamic analysis tool), **software frameworks** (imagine a data analytics infrastructure), and **software institutes** (hubs of excellence in software engineering).

Proposals focusing purely on computer science can be funded through the following divisions of the Computer and Information Science and Engineering (CISE) directorate: Advanced Cyberinfrastructure (ACI), Computing and Communication Foundations (CCF), Computer and Network Systems (CNS), and Information & Intelligent Systems (IIS). Furthermore, cross-disciplinary proposals can also be funded through the following directorates: Biological Sciences (BIO); Education and Human Resources (EHR); Engineering (ENG); Geosciences (GEO); Mathematical and Physical Sciences (MPS); Social, Behavioral & Economic Sciences (SBE).

In the USA, software research is also funded by the Defense Advanced Research Projects Agency (DARPA), which is responsible for the military's development of emerging technologies. According to information that is publicly available,[4] DARPA is sponsoring diverse software engineering research projects, on topics such as crowd-sourced formal verification (CSFV), long-lived software systems (Building Resource Adaptive Software Systems — BRASS), and the automatic extraction of reusable components from open source software (Mining and Understanding Software Enclaves — MUSE). Results of DARPA-funded software research have led to significant contributions of important industrial relevance. Widely-used examples include the **Coverity** static analysis tool, the **Valgrind** dynamic analysis tool, and Security-Enhanced Linux (**SELinux**).

CHINA

State-sponsored software engineering research in China for the period 2016–2020 is directed through the country's Thirteenth Five-Year Plan. This focuses on cloud computing, big data infrastructures, data-driven software, intelligent systems, cognitive perception, and human-computer interaction. Specifically, the program's four pillars comprise the following tasks.

[3] http://www.nsf.gov/funding/pgm_summ.jsp?pims_id=504730

[4] http://www.darpa.mil/tag-list?tt=52

- **Cloud computing and big data infrastructure**: foundational data science theory; new generation cloud computing technologies and devices; new storage technologies and big data platforms; highly efficient cloud computing data center technologies; big data management systems.
- **New data-driven software based on the cloud model**: theory, methods and techniques for intelligent software; software architecture and supporting technologies for intelligent systems; software development methodologies and environments for crowd-intelligence; domain specific applications; cloud computing and big data open source community ecosystems.
- **Big data analytics applications and human-like intelligence**: foundational theory and techniques for big data analytics; new intelligent processing computer architectures; highly efficient, scalable computing architecture models and optimization techniques; big data-driven humanoid intelligent cognition and intelligent decision; large-scale visual computing technologies and key applications; domain specific applications.
- **Cloud integration cognitive perception and human-computer interaction:** computable models of human behaviour and natural interaction theory; multi-source data-driven intelligent and efficient modelling; technology, equipment and interface tools supporting cloud integration interaction; new applications supporting cloud computing interaction.

INDIA

Government funding for software engineering research in India is limited. It comes from departments, ministries, the Science and Engineering Research Board. In addition, publicly funded research happens through the institutes overseen by the Council of Scientific & Industrial Research (CSIR), the Indian Council of Agricultural Research (ICAR), the Indian Council of Medical Research (ICMR), the Indian Space Research Organization (ISRO), and the Indian Council of Social Science Research (ICSSR).

A few companies such as TFIR, Infosys, TCS, and IBM fund academic software engineering research (both in India and abroad) through targeted projects, alliances, university-based research parks, incubators, research agreements, and exchanges of personnel. In addition, all major software development companies fund their own internal research projects. It appears that numerous companies are focusing their research on **software analytics** and corresponding **business platforms**.

CONCLUDING REMARKS

Significant focused investment in software engineering research can help Europe stay on top and even lead a world that is increasingly defined and shaped by software. The need for targeted research in software engineering is prompted by developments in three broad areas. First, the computing landscape is changing from top to bottom. Cloud computing and infrastructure as code allow the radical rethinking of computing systems; universal memory obsoletes current memory hierarchies; multicore architectures change how energy efficiency and performance are to be obtained; while big data, machine learning, and natural user interfaces open new possibilities for computing applications. Software lies at the heart of all these changes. Second, seven software-driven vertical application domains are reshaping entire industries and society as a whole. These domains are autonomous vehicles, massive open online courses, open intellectual property, the Internet of Things, life sciences, 3D printing, financial technology, and Industry 4.0. Third, the computing landscape's technological trends and the changes in the vertical application domains, give rise to several critical crosscutting software engineering challenges. These challenges involve the taming of the immense scale and complexity of the required software through suitable tools and abstractions, data analytics, novel processes and collaboration mechanisms, as well as the integration of software development with operations. On top of these, return with a vengeance known challenges regarding security, privacy, reliability, and environmental sustainability.

The forces outlined in the preceding paragraph are interrelated. Changes in the computing landscape enable development in new vertical domains, which in turn fund sophisticated computing systems and infrastructures. Both types of changes establish cross-cutting challenges. These challenges can in turn be sometimes addressed through revolutionary technologies (consider static program analysis or public key cryptography) which can subsequently drive new vertical application domains, tying the relationships in a full circle.

The software research that should be performed is informed by the specific challenges. It covers areas where past advances point to new victories we can expect in the future. Three concrete priorities emerge from this study, mainly as a result of the relentlessly increasing complexity, size, and importance of software-intensive systems. The corresponding targeting of research investment can drive progress in both the software industry and in the domains that increasingly depend on software.

1. **Software Construction** Although no scientific advance will ever deliver a silver bullet, investment in technologies and tools that support software construction will increase its effectiveness. This includes the development of new abstraction methods, construction tools, components, and frameworks, as well as the empirical evaluation of existing and new approaches. Particular emphasis should be placed on approaches that improve quality or expand the reach of software engineering into new domains.

2. **Software Design** Given the challenges outlined in preceding sections, this priority's goals should be the democratization of software design through packaging and reusability. This involves creating high-quality designs that address key challenges, such as concurrency, data persistence, distribution of components, security, reliability, and user interaction, in a way that can be readily packaged and easily reused by diverse software systems.
3. **Software Engineering Process** The spread of web-based platforms, tools, and open source software projects has allowed even small organizations to benefit from sophisticated software engineering processes. This provides a fertile ground for new research contributions to flourish and accelerate the software industry's growth. Important areas include the integration of requirement elicitation and maintenance into agile processes as well as the holistic management of software development with IT operations (DevOps). A key element of such research should be its empirical evaluation in large-scale realistic settings.

Software engineering research as a whole must be funded as a specific priority, so that it can act as a foundation for the robust evolution of computing and its applications. The findings of such research, whether empirical, such as the effect of size or reviews on software quality, or technological, such as static program analysis tools and model checkers, increase the IT industry's efficiency and benefit society through more and higher quality software. In contrast, restricting software engineering research to the incidental support that takes place within the development of applications, is like trying to build a national highway network by patching together cobblestone paths of small villages. Such an approach is ineffective and drags down the economy.

In the coming decade software will affect crucial aspects of modern society: jobs, production, and the nature of work; security, privacy, transparency, and trust in the physical and cyber domains; government and social structures; as well as the ownership of tangible and virtual goods. Investing in the software engineering research required to progress in these areas can help Europe lead in the corresponding changes, while failure to commit resources may severely limit its say and options. Given the breadth of the upcoming changes, the proposed investment can spur a stronger economy, higher-quality jobs, increased security, better governance, and greener living.

Europe can build on its world-class pockets of excellence in specific areas of economic activity, such as automotive manufacturing, engineering, aerospace technology, financial services, and luxury goods marketing, by responding to competition from other economic powerhouses and addressing shortcomings in the amount of funding and its processes. Thus, focused, significant, and effective research funding in the area of software engineering will enable the development of new methods, tools, architectures, systems, business models, processes, and applications that can be instrumental in the establishment of Europe as the center of the rising software-run economy.

UNTANGLING THE ALPHABET SOUP

CMOS: Complementary metal–oxide–semiconductor

DARPA: Defense Advanced Research Projects Agency (USA)

DevOps: Development and Operations

DFG: Deutsche Forschungsgemeinschaft — Germany's research funding body

DRAM: Dynamic random access memory

EPSRC: Engineering and Physical Sciences Research Council (United Kingdom)

GPU: Graphics processing unit

GPGPU: General purpose graphics processing unit

HPC: High performance computing

IaaS: Infrastructure as a service

IaC: Infrastructure as code

ICT: information and communication technology

IP: Intellectual property

IoT: Internet of things

LS: Life sciences

ML: Machine learning

MOOC: Massive open online course

NSF: National Science Foundation (USA)

NUI: Natural user interface

NVM: non-volatile memory

PaaS: Platform as a service

PCM: phase change memory

PSD: Payment Services Directive

QC: Quantum computing

SaaS: Software as a service

SDDC: Software-defined data center

SDN: Software-defined networking

SDS: Software-defined storage

SDx: Software-defined anything

SSD: Solid state drive

STT-RAM: spin-transfer torque RAM

ACKNOWLEDGEMENTS AND DISCLAIMER

The author of this report would like to thank the following individuals for informally providing valuable input regarding the future directions of software engineering research: Ayse Bener, Brian Brannon, Anita Carleton, Jeff Carver, Alexandros Chatzigeorgiou, Jane Cleland-Huang, Marios Fragkoulis, George Giaglis, Sol Greenspan, Daniel Katz, Phillip Laplante, Xabier Larrucea, Casper Lassenius, Panos Louridas, Henry Muccini, Mei Nagappan, Zeljko Obrenovic, Ipek Ozkaya, Cesare Pautasso, Rafael Prikladnicki, Rajiv Ramnath, Walker Royce, Richard Selby, Tushar Sharma, Forrest Shull, Jens Weber, Adam Welc, Laurie Williams, and Minghui Zhou. The input from the participants in the workshop "Challenges & Opportunities for the European Software Industry", Pearse O' Donohue, Katrin Schleife, Bernd Beckert, Mike Hinchey, Sotiris Koussouris, Lutz Schubert, and Francisco Medeiros, is also gratefully acknowledged. All views and opinions expressed in this report are those of its author and are not necessarily shared by any of the listed individuals, their organizations, the European Commission, or the Athens University of Economics and Business.

BIBLIOGRAPHY AND REFERENCES

[ACFD+14] Iván Arce, Kathleen Clark-Fisher, Neil Daswani, Jim DelGrosso, Danny Dhillon, Christoph Kern, Tadayoshi Kohno, Carl Landwehr, Gary McGraw, Brook Schoenfield, Margo Seltzer, Diomidis Spinellis, Izar Tarandach, and Jacob West. Avoiding the top 10 software security design flaws. Technical report, IEEE Computer Society, Center for Secure Design, August 2014.

[AFF+14] H. Alkhatib, P. Faraboschi, E. Frachtenberg, H. Kasahara, D. Lange, P. Laplante, A. Merchant, D. Milojicic, and K. Schwan. IEEE CS 2022 report. Available online http://www.computer.org/cms/Computer.org/ComputingNow/2022Report.pdf , 2014.

[AFF+15] H. Alkhatib, P. Faraboschi, E. Frachtenberg, H. Kasahara, D. Lange, P. Laplante, A. Merchant, D. Milojicic, and K. Schwan. What will 2022 look like? the IEEE CS 2022 report. *Computer*, 48(3):68–76, Mar 2015.

[AKN+16] Paris Avgeriou, Philippe Kruchten, Robert L. Nord, Ipek Ozkaya, and Carolyn Seaman. Reducing friction in software development. *Software, IEEE*, 33(1):66–73, Jan 2016.

[BBC+16] Christoph Becker, Stefanie Betz, Ruzanna Chitchyan, Leticia Duboc, Steve M. Easterbrook, Birgit Penzenstadler, Norbet Seyff, and Colin C. Venters. Requirements: The key to sustainability. *Software, IEEE*, 33(1):56– 65, Jan 2016.

[BLBM15] Giancarlo Bruno, Abel Lee, Matthew Blake, and Jesse McWaters. The future of financial services. Technical report, World Economic Forum, June 2015.

[BM14] Margaret M. Burnett and Brad A. Myers. Future of end-user software engineering: Beyond the silos. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 201– 211, New York, NY, USA, 2014. ACM.

[Bos16] Jan Bosch. Speed, data, and ecosystems: The future of software engineering. *Software, IEEE*, 33(1):82–88, Jan 2016.

[BZ14] Andrew Begel and Thomas Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 12–23, New York, NY, USA, 2014. ACM.

[CHGHH+14] Jane Cleland-Huang, Orlena C. Z. Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software traceability: Trends and future directions. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 55–69, New York, NY, USA, 2014. ACM.

[CSSR14] Satish Chandra, Vibha Singhal Sinha, Saurabh Sinha, and Krishna Ratakonda. Software services: A research roadmap. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software*

Engineering Track, FOSE 2014, pages 40–54, New York, NY, USA, 2014. ACM.

[DSS+14] Balakrishnan Dasarathy, Kevin Sullivan, Douglas C. Schmidt, Douglas H. Fisher, and Adam Porter. The past, present, and future of MOOCsand their relevance to software engineering. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 212– 224, New York, NY, USA, 2014. ACM.

[Ebe15] C. Ebert. Looking into the future. *Software, IEEE*, 32(6):92–97, Nov 2015.

[Esp15] Victoria Espinel. Deep shift: 21 ways software will transform global society. Survey report, World Economic Forum, November 2015.

[FDN14] Alfonso Fuggetta and Elisabetta Di Nitto. Software process. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 1–12, New York, NY, USA, 2014. ACM.

[Gar14] David Garlan. Software architecture: A travelogue. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 29–39, New York, NY, USA, 2014. ACM.

[GHNR14] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 167– 181, New York, NY, USA, 2014. ACM.

[HBD+14] Richard Hudson, Kristian Borch, Stephanie Daimer, David Charles De Roure, Kurt Deketelaere, Apostolos Dimitropoulos, Ulrike Felt, Aldo Geuna, Jerome Glenn, Krzysztof Gulda, Jana Kolar, Jordi Molas Gallart, Rajneesh Narula, Gill Ringland, Petra Schaper-Rinkel, John Smith, Anna Tschaut, and Marijk van der Wen. The knowledge future: Intelligent policy choices for europe 2050. Available online https://ec.europa.eu/research/foresight/pdf/knowledge_future_2050.pdf, 2014. Report by an expert group on Foresight on Key Long-term Transformations of European systems: Research, Innovation and Higher Education (KT2050).

[Her14] *FOSE 2014: Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, New York, NY, USA, 2014. ACM.

[HJP16] Emily Hill, Philip M. Johnson, and Daniel Port. Is an athletic approach the future of software engineering education? *Software, IEEE*, 33(1):97–100, Jan 2016.

[HKB16] James Herbsleb, Christian Kastner, and Christopher Bogart. Intelligently transparent software ecosystems. *Software, IEEE*, 33(1):89–96, Jan 2016.

[HWK+14] John Hatcliff, Alan Wassyng, Tim Kelly, Cyrille Comar, and Paul Jones. Certifiably safe software-dependent systems: Challenges and directions. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 182–200, New York, NY, USA, 2014. ACM.

[LvdH16] Thomas D. LaToza and Andre van der Hoek. Crowdsourcing in software engineering: Models, motivations, and challenges. *Software, IEEE*, 33(1):74–80, Jan 2016.

[MFP16] Claudia de O. Melo, Ronaldo Ferraz, and Rebecca J. Parsons. Brazil and the emerging future of software engineering. *Software, IEEE*, 33(1):45–47, Jan 2016.

[MHG14] Emerson Murphy-Hill and Dan Grossman. How programming languages will co-evolve with software engineering: A bright decade ahead. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 145–154, New York, NY, USA, 2014. ACM.

[MNJR16] Walid Maalej, Maleknaz Nayebi, Timo Johann, and Guenther Ruhe. Toward data-driven requirements engineering. *Software, IEEE*, 33(1):48–54, Jan 2016.

[Moc14] Audris Mockus. Engineering big data solutions. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 85–99, New York, NY, USA, 2014. ACM.

[MONF16] Andrew Moore, Tim O'Reilly, Paul D. Nielsen, and Kevin Fall. Four thought leaders on where the industry is headed. *Software, IEEE*, 33(1):36–39, Jan 2016.

[MP14] Andreas Metzger and Klaus Pohl. Software product line engineering and variability management: Achievements and challenges. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 70–84, New York, NY, USA, 2014. ACM.

[OR14] Alessandro Orso and Gregg Rothermel. Software testing: A research travelogue (2000–2014). In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 117–132, New York, NY, USA, 2014. ACM.

[Pat10] David Patterson. The trouble with multi-core. *IEEE Spectrum*, 47(7):28–32, July 2010.

[PJM+14] Gian Pietro Picco, Christine Julien, Amy L. Murphy, Mirco Musolesi, and Gruia-Catalin Roman. Software engineering for mobility: Reflecting on the past, peering into the future. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 13– 28, New York, NY, USA, 2014. ACM.

[Raj14] Václav Rajlich. Software evolution and maintenance. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 133–144, New York, NY, USA, 2014. ACM.

[SCC+16] Forrest Shull, Anita Carleton, Jeromy Carriere, Rafael Prikladnicki, and Dongmei Zhang. The future of software engineering. *Software, IEEE*, 33(1):32–35, Jan 2016.

[Spi07] Diomidis Spinellis. Cracking software reuse. *IEEE Software*, 24(1):12–13, January/February 2007.

[Spi15] Diomidis Spinellis. Extending our field's reach. *IEEE Software*, 32(6):4–6, Nov/Dec 2015.

[SSC+14] Margaret-Anne Storey, Leif Singer, Brendan Cleary, Fernando Figueira Filho, and Alexey Zagalsky. The (r)evolution of social media in software engineering. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 100–116, New York, NY, USA, 2014. ACM.

[TYXL16] Zhengrong Tang, Melissa Yang, Joshua Xiang, and John Liu. The future of Chinese software development. *Software, IEEE*, 33(1):40–44, Jan 2016.

[VBS14] Willem Visser, Nikolaj Bjørner, and Natarajan Shankar. Software engineering and automated deduction. In *Proceedings of the 36th International Conference on Software Engineering — Future of Software Engineering Track*, FOSE 2014, pages 155– 166, New York, NY, USA, 2014. ACM.