Software Process

Alfonso Fuggetta
Politecnico di Milano and CEFRIEL
Via Fucini, 2
20133 Milano - Italy
alfonso.fuggetta@polimi.it

Elisabetta Di Nitto
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 Milano - Italy
elisabetta.dinitto@polimi.it

ABSTRACT

This paper is a travelogue of Software Process research and practice in the past 15 years. It is based on the paper written by one of the authors for the FOSE Track at ICSE 2000. Since then, the landscape of Software Process research has significantly evolved: technological breakthroughs and market disruptions have defined new and complex challenges for Software Engineering researchers and practitioners.

In this paper we provide an overview of the current status of research and practice, highlight new challenges, and provide a non-exhaustive list of research issues that, in our view, need to be tackled by future research work.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*software process models, life cycle, programming teams*

General Terms

Management, Performance, Human Factors.

Keywords

Software Process, Software Development, Agile Software Development, Software Development Environments, Social Factors in Software Development, Empirical Studies.

1. INTRODUCTION

Year after year, software has become an increasingly essential and vital constituent of our society. There is no business sector or aspect of our daily life that is not affected by software. Personal and work activities, business and economic initiatives, civil and industrial infrastructures, politics, education, and entertainment – just to name a few – are all **deeply permeated and governed by software applications and systems**. Consequently, software development has become a critical activity that needs to be carefully studied, understood, improved, and supported.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

FOSE'14, May 31 – June 7, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2865-4/14/05...$15.00
http://dx.doi.org/10.1145/2593882.2593883

Over the past half of a century, the goal of the Software Engineering community has been to study and tackle these issues and problems. Within this community, **Software Process research** has focused on understanding, describing, evaluating, automating, and improving the procedures, policies, and techniques used to master this complex endeavor. This research work has produced many interesting results. At the same time, however, many goals and objectives have been ill-framed or overemphasized, while others have been overlooked. Certainly, *many problems are still in search of convincing approaches able to thoroughly address and tackle them.*

The term Software Process gained major visibility in the eighties and since then has sparked a lot of projects and initiatives in different areas: modeling, automation, improvement to name a few. At ICSE 2000, one of the authors wrote a paper about the achievements, mistakes, and open challenges of this research area [18]. The paper was presented in the FOSE (Future of Software Engineering) Track. Since then, a lot has happened. The Internet has redefined all aspects of our society. Mobile devices have radically changed the way citizens and companies approach Information and Communication Technologies (ICTs). Totally new sectors and markets have been created from scratch (e.g., social networks). Existing approaches to software development and software distribution have been completely transformed by the advent and wide diffusion of concepts such as open source software development, agile methods, and development platforms for mobile applications. Consequently, it does make sense to reconsider the state of the Software Process research community, to critically revise the way it has dealt with the issues and opportunities identified in year 2000 and, at the same time, to discuss how it is coping with the new challenges and deep transformations that have occurred over the past 15 years.

This paper has the ambition to discuss these issues and to provide some contributions for the evolution and strategic development of the community. To pursue this goal, the paper is organized as follows:

- Section 2 summarizes the main comments proposed in the FOSE 2000 Paper.
- Section 3 provides a quick overview of the main trends and directions in software process research over the past decade.
- Section 4 highlights some key challenges and needs emerging from industries, practitioners, and the market in general.

- Section 5 proposes some directions and suggestions for future research.
- Section 6 presents some conclusions and closing remarks.

2. SOFTWARE PROCESS IN YEAR 2000

The Year 2000 marked an important milestone in Software Engineering research. Increasingly, many researchers and practitioners questioned themselves about the value and impact of the results achieved in the past decades. Of course, the advent of the new century introduced a particular emphasis and a unique symbolism. However, there was also an increasing perception that the research work in many areas of Software Engineering had reached a sort of turning point and needed to be rethought and redefined. Were the results really able to influence and address the issues and problems experienced by software developers, or were they too often an intellectual exercise unable to concretely (and proactively) affect the way software is conceived, developed, deployed, used, and managed in real and practical scenarios?

As far as the trends in Software Process research are concerned, the analysis accomplished in the FOSE 2000 paper identified four main areas of investigation:

Process modeling and support A software process is a complex endeavor involving professionals, organizations, company policies, tools & support environments. A first research area was centered on the creation of languages and environments to describe and "model" such complex processes. The resulting models had to be precise enough to facilitate the comprehension and analysis of the process, and to provide automated support to developers and managers involved in its "enactment". This research stream generated many Process Modeling Languages (PMLs) and associated run-time infrastructures. Such languages were based on a variety of paradigms (logic programming, finite state automata, Petri Nets, ...) and had to deal with complex issues such as inconsistency management and integration of developers' different viewpoints.

Process improvement A second important research area was process improvement. Software development activities are intrinsically dynamic and continuously evolving entities that cannot be frozen or defined once for all. It is therefore essential to identify methods and approaches to study the maturity of a process and to identify strategies and procedures to improve it.

Metrics and empirical studies Software development is a complex activity involving technical, human, and social/organizational factors. For this reason, it is not easy to study and assess its behavior and determine its performance. During the nineties, many researchers developed and applied techniques and methods to perform empirical studies and identify useful metrics able to characterize and assess software processes performance.

"Real" Processes! Finally, based on the experiences and achievements of the previous decades, several "concrete" processes were conceived and applied, e.g., the Unified Process and the Personal Software Process.

Based on this substantial amount of research, there were several attempts to assess the quality, significance, and areas of improvement of the work accomplished in those years (see for instance [2, 12, 34]). The FOSE 2000 paper analysed these attempts and elaborated a number of considerations and criticisms:

1. *"Software processes are processes too".* Rephrasing the famous paper title by Lee Osterweil [24], it is important to observe that software processes are first and foremost "processes", i.e., **it is not possible (and useful) to consider software processes as specific and unique entities that deserve a completely autonomous and independent research effort**. Certainly, there are a few specific challenges that qualify uniquely and distinctively the software domain, but there are also many commonalities and synergies with the work accomplished in other research communities (e.g., workflow management), that in the past have been largely overlooked.
2. *"The purpose of process languages and technology must be rethought".* Many languages developed during the 90s were **too complex and rigid**. In general, they turned out to be difficult to apply in real settings and not capable of producing a concrete impact on software development practices. Certainly, it was (*and still is*) important to provide means to describe processes effectively and also to provide automation whenever this is possible and meaningful (e.g., configuration management). But PMLs went too far and basically "missed the point".
3. *"Empirical studies are a means not an end".* As with PMLs, the risk of many empirical studies was to become formal exercises without practical and concrete relevance. Certainly, from a methodological viewpoint it has been essential to introduce structure and rigor in assessing the performance and behavior of software processes, but too often the "internal validity" and quality of empirical studies (i.e., their intrinsic and methodological quality) have outshone their external validity, i.e., their ability to provide general and universally usable insights and results.
4. *"Software process improvement is process improvement too".* Similarly to what happened in software support and automation research, process improvement suffered from a sort of "isolation syndrome": too often software processes were considered a specific and unique category, thus ignoring or largely underestimating the contributions of other disciplines and research areas. Indeed, improving software development processes requires most of (if not all) the techniques and methods used for any other complex people-centered activity.

These criticisms raised a significant debate at the conference and in general in the community. Indeed, they reflected the feelings and moods of many researchers. Most important, they were based on a crude and sincere analysis of the results of the research work accomplished in those years. While there were several useful and successful achievements, significant parts of that research work turned out to be ineffective and largely ignored.

The dramatic and incredibly fast development of many enabling technologies and practices (mobile internet devices, open source software, ...) has further reinforced the idea that software process research definitely needed a significant change in scope and approach. Over the years, the community has understood the need for such change and has produced the important results that we discuss in the next section.

3. SOFTWARE PROCESS IN THE PAST DECADE

In the past decade, research on software processes has evolved significantly. Some of the concerns discussed in the FOSE 2000 paper have been at least partially addressed. In particular, it has been widely recognized that software process is a multidisciplinary research domain and, consequently, it should consider and incorporate contributions from other disciplines. For instance, the main process improvement approach, CMM, has evolved into the Capability Maturity Model Integration (CMMI) framework[1]. CMMI extends CMM by offering a finer grained mechanism for measuring performance of an organization and by including specific *process areas*, where organizational factors have a key role.

Nowadays researchers are much less concerned about modeling and executing processes. They have recognized that the possibilities of fully automating them are limited to specific aspects and phases such as code generation, testing, packaging, deployment and operation management of final products. Moreover, they have agreed that software process performance is heavily influenced by and structured around the role and behavior of individuals and organizations [38].

To characterize this shift in focus, this section provides a quick overview of the most relevant directions in software process. The goal is not to provide an in-depth analysis and evaluation of the results being accomplished: rather, the aim is to highlight important topics and to contrast them with the needs and challenges emerging from the IT market and the society in general.

3.1 Social Aspects

Over the years, the literature on software processes has been increasingly recognizing the importance of social aspects in software processes. For instance, Tamburri and others [32] have analyzed the literature on organizational social structures to identify those that are suitable to be exploited in the software engineering domain.

The software development process has been considered a *socio-technical system*, where organizational and human aspects have a key role and have to be supported by technology in a way that is human and organization-driven. The term socio-technical system has been first used by Eric Trist, Ken Bamforth and Fred Emery in the fifthies, based on their studies of workers in coal mines [33]. In particular, they noted that the introduction of technology does not necessarily result in improved performance. Improvement is the result of proper interplay between 1) humans-centered aspects – in particular, social aspects – and 2) technology. Thus, these two elements should not be studied and optimized in isolation, as this approach typically fails to achieve the desired objective.

[1]`http://cmmiinstitute.com`

By interpreting the concept of socio-technical systems within the context of software processes, Cataldo and others [8] have identified *socio-technical congruence* as a way to define the fit between product dependencies, the resulting task dependences, and the actual coordination activities occurring during the development process. As argued by Blincoe and others [5, 4], a timely identification of coordination requirements in terms of task dependencies becomes an important factor to improve the performance of the software development process.

3.2 Agile Processes

The importance of social aspects in the development process has been also the driving factor that sparked the creation of the *Agile Manifesto*[2]. The Manifesto is based on some key principles:

1. the software product is the essential focus of the development process;
2. human behavior and the quality of interactions are essential enablers and success factors;
3. incremental and spiral approaches, based on frequent releases and strict collaboration with the customer, are the quintessential traits of modern software development initiatives;
4. in general, it is vital to promptly and quickly react to requirement changes, at any stage of the development process.

The Agile Manifesto has spawned many approaches and methods. Among the others, Scrum is certainly extremely popular and successful [15]. According to this method, the process is organized in *sprints*, i.e., development iterations of at most 4 weeks each. Every iteration is focused on the development of a specific set of features. Project tracking and control are achieved through short daily meetings in which each team member describes what he/she has been doing the day before, what he/she is going to do in the current day, and if there is any important issue to discuss. Such meetings help identifying issues very early and changing plans accordingly if needed. Changes in user requirements are accepted and welcomed, but they are not considered in the current sprint. They are described and prioritized so they can be considered in the forthcoming sprints. At the end of each sprint, the features that were the focus of the sprint have to be fully completed, i.e., not only they should be implemented, but also verified and documented, so that they can be shipped to the customer.

Several studies have shown the advantages of Scrum in a variety of situations and contexts. Among the others, it is worthwhile mentioning the experiences in the application of Scrum at Google [28]. Starting from a fairly immature and chaotic situation (in the CMMI terminology), the application of Scrum allowed engineering teams to achieve a better control on the development process, a more careful prioritization of features, and more realistic deadlines. Another interesting experience has been reported by Systematic Software Engineering (SSE). SSE is a very mature organization (level 5 in the CMMI scale) that has adopted a Scrum-based approach for its software development processes [29]. This

[2]`http://agilemanifesto.org`

study confutes the critical remarks of many detractors of agile approaches according to which agile methods lack structure and organization, and therefore are difficult to apply to mature and large software development processes.

3.3 Global Software Engineering

The need for coordination and agility becomes certainly more challenging in a Global Software Engineering (GSE) setting, where the software process is enacted by teams that are distributed across different countries and time zones [19]. The motivation for distributing development teams goes beyond issues related specifically to software development activities, and may concern a number of aspects and needs:

- establish and maintain a direct and strong contact with customers at remote locations;
- exploit the availability of remote workers;
- reduce costs by offshoring part of the process to a remote subsidiary/company;
- enhance the capabilities of small companies by creating networks and cooperation frameworks with other development organizations and teams.

For these reasons, researchers have recognized the importance of promoting GSE practices. They have analyzed open problems and emerging issues, and have coherently identified new tools and methods to address them. In particular, the literature offers interesting studies focusing on the applicability to GSE of various process paradigms, and in particular of Scrum [11]. Other studies discuss issues and topics such as the impact of architectural choices on the development process [10] and the value of exploiting innovative tools to support awareness [21], or report specific success stories of cooperation either within large companies or among SMEs [31]. Unfortunately, despite the large amount of work, so far it is still difficult to draw general lessons and conclusions from the cases that have been proposed by the research community.

In general, the experiences of different authors at different organizations suggest that success and failure are driven primarily by the ability to establish effective mutual relations and trust among software developers. Whether and how we can systematically promote them in a GSE-specific context is still to be understood [19].

3.4 The Role of Empirical Software Engineering

Over the past decade, empirical studies in software engineering have significantly evolved, both in terms of quality of the methods being used, and in terms of the external validity of the results that have been proposed. Indeed, there have been several contributions that have focused on the characterization and evaluation methodology and approach used in empirical studies (see for instance [36]).

Currently, there are many ongoing research works centered on rigorous empirical studies whose goal is to show the advantages of some new approach/practice [25], or to demonstrate the validity of a research hypothesis [7]. In addition, while many older studies were negatively affected by the impossibility to replicate them, the tremendous growth of open source projects has enriched the amount of data and observation viewpoints available to researchers. This has made it possible to apply empirical analysis on a wider scale, thus resulting in a significant improvement of the quality of the studies and of the related findings.

3.5 Model-Driven Engineering

Model-driven engineering is a specific approach to software development that focuses on using models as the archetypes to build software systems. This enables traceability from requirements to code, (automatic) generation of code for different target platforms, and analysis of models to predict the ability of the final system to fulfill specific QoS characteristics.

The main reference for model-driven development is the OMG specification named Model-Driven Architecture (MDA) [23], which has been released in an intermediate version (1.0.1) in 2003 and was planned to be replaced by a new one by 2005. So far, the new release has not been issued yet, but the work around MDA is progressing, as witnessed by the presence of specific conferences and workshops on this subject (most notably, MODELS 2013).

The MDA approach identifies three different viewpoints used to describe a system: the *Computation Independent Model (CIM)*, the *Platform Independent Model (PIM)*, and the *Platform Specific Model (PSM)*. CIMs usually represent concepts specific of a given business. This level usually describes requirements, enterprise architectures, and business models. PIMs discuss how the system works by abstracting the platform-specific elements. At this level, the architectural models are elaborated describing the business logic of an application. PSMs introduce aspects that are concerned with the usage of a specific platform (e.g., JEE) for the implementation. The underlying idea is that, given the definition of the target platform within a *Platform Model*, a PIM model can be automatically transformed into a PSM one, suitable for direct code generation.

Even if full adoption of model-driven engineering by industry is still limited, the scientific literature reports some interesting case studies. Among the others, the recent work by Briand et al. [6] provides useful hints on how to promote the adoption of a model-driven approach in real industrial contexts. In particular, it describes three specific cases of adoption that are particularly interesting as they have not been achieved within IT companies, but in organizations focusing on the maritime or energy sector. One of the reported cases refers to the adoption of a model-driven approach to support a software safety certification process, by ensuring traceability of safety requirements. Another interesting paper focusing on the adoption of MDA in industry is by Whittle and others [35]. They highlight the importance of establishing processes within the companies to support the adoption of MDA. Moreover, they argue about the importance of matching tools to the people who are going to use them. In this respect, their conclusions are in line with the ones proposed in [6].

3.6 ALM

Application Lifecycle Management (ALM) suites are a class of products originated partly by the conventional software industry and partly by the open source community. Even though they do not appear as explicitly connected to Software Process research, they offer mechanisms for automating some tasks (typically, build and test), and for connecting managerial tasks with software development ac-

tivities. Moreover, they offer connectors to external tools for specific functionality such us configuration management, bug tracking, requirement management and the like. Examples of ALM tools are Microsoft Visual Studio Application Lifecycle Management[3], IBM Rational solutions for CLM[4], and the open source suite MyLyn[5], which is integrated in Eclipse and has then evolved into the Tasktop commercial product[6].

In [9] Application Lifecycle Management is presented as a discipline aiming at taking care also of application operation procedures and managerial governance. However, as the author claims, current ALM tools do not support the integration of these aspects with those related to software development.

As we will see in the next section, the integration of software development into a broader context is also the objective of the *DevOps* movement.

3.7 Where Automation Really Matters

Nowadays, the most important application of automation in software processes is in supporting the final phases of software development.

3.7.1 Configuration Management

Configuration/version management has been revitalized by new tools, most notably, git[7], but also by software forges, i.e., collaboration platforms, typically made available as Software-as-a-Service (SaaS), which are useful to organize and make visible to developers and external observers many pieces of information concerning a software being developed. Thanks to forges, version management is not seen anymore as a very difficult feature to install and operate, but as a service that teams use daily for managing software, conducting discussions, sharing information within the project team.

3.7.2 Quality Assurance

Automatic quality assurance tools and processes have reached a significant level of usability, and are increasingly adopted and used by software development teams and organizations.

- Developers can easily use source code analysis tools that provide indicators and metrics about the quality of the software being developed. An example of these tools is SonarQube[8].
- Testing can be significantly automated. The execution of unit and integration test is now already largely automated thanks to the diffusion of the XUnit family. Another significant example of tool in this area is Selenium[9], which supports the creation of automated test cases for a web application by recording testing scripts from the actions of a user on the corresponding web interface.
- Acceptance-level tests can be automated as well using tools such as Easyb[10] or Fitnesse[11]. They make it possible to rigorously express requirements and to automate test execution. For instance, Easyb uses a Domain Specific Language that enables the association of scenarios with the execution of specific pieces of code, while in Fitnesse requirements are expressed in terms of tables that extensively define the correspondence between inputs and associated outputs.
- Finally, even the testing process as a whole is being automated. For instance, the STAF framework[12] offers a quite complete, even though textual (XML-based), scripting language for defining the testing process. STAF makes it possible to model when testing should be triggered, which tests should be executed, and under which circumstances.

3.7.3 Software Building

Automation is possible and exploited in all the activities and processes related to building, deploying, and operating software systems. Let's consider here a few significant examples:

- By running a Maven[13] script, every user is able to check out a software release from a version management system, automatically download all needed external components and libraries, and get ready for the deployment of the final product.
- Frameworks like Jenkins[26] close the *develop, integrate, check quality, build* cycle by integrating some of the previously mentioned tools, and by offering an environment that can even be configured to automatically generate the *ready to be deployed product*, as soon as a new version of a single source code file is pushed into the version management system. This increases the awareness of the development team about the impacts and results of the development activity. This approach is known as *continuous integration* [17].
- Automation plays a major role also in the operation phase, as far as the management of applications and their related software stack is concerned. This has been enabled by the introduction of virtualization and remote control mechanisms and, more recently, the diffusion of public and private clouds. For instance, Puppet [14] is an example of a tool supporting application management, while Nagios [15] can be used for monitoring the application state.

3.7.4 DevOps

The increasing level of automation enables software developers to fulfill the requirements of business critical applications in terms of availability, reliability and response time. At the same time, it creates a bridge between the development and the operation activities that is often identified by the term *DevOps* [14]. DevOps is determining a significant

[3] http://msdn.microsoft.com/library/fda2bad5(VS.100).aspx
[4] http://pic.dhe.ibm.com/infocenter/clmhelp/v4r0m1/index.jsp
[5] http://www.eclipse.org/mylyn/
[6] https://tasktop.com
[7] http://git-scm.com
[8] http://www.sonarqube.org
[9] http://docs.seleniumhq.org
[10] http://easyb.org
[11] http://fitnesse.org
[12] http://staf.sourceforge.net
[13] http://maven.apache.org
[14] http://puppetlabs.com
[15] http://www.nagios.org

organizational change within IT companies. For instance, as reported in [20], organizations like Amazon are choosing a "per service" organization, where cross-functional teams are entirely responsible for the development and the operation of a set of services. The metrics used for assessing the performance of such teams are focused on the quality and stability of the final services, rather than on the productivity measurements traditionally suggested by software engineering research.

4. MAJOR TRENDS AND CHALLENGES

The previous sections have briefly summarized the achievements, issues, and research directions that have been characterizing the Software Process research community in the past two decades. Looking ahead, to understand and imagine the future of Software Process research, it is vital to discuss the trends that are reshaping the IT market as a central economic sector of our society and, consequently, Software Engineering as an essential and critical scientific discipline. In our view, these trends can be structured around four main major challenges.

4.1 Challenge 1: The Internet is the Development Environment

Nowadays, **the Internet is the development environment**, i.e., any development activity is carried out over the Internet, even for small scale developments.

1. We collaborate and work over the Internet, within a single company or across the boundaries of different organizations or with sparse and distributed individuals. Notions such as "development environment" and "development team" have radically and dramatically changed: software is rarely developed in isolation, as it is more and more the result of interaction, integration, and cooperation among developers and between developers and end-users working in a Networked Software Development context [30]. This is driving a **profound and radical change in the methods and techniques used to conceive, design, develop, test, deploy, and evolve software**. Crowdsourcing is an example of this change that still needs to be fully understood and evaluated [37].

2. Software is continuously changed and redeployed based on customers' requests and expectations. This poses huge challenges to classical processes such as configuration management, and software deployment and operation. The corresponding environments and support technologies have to **reliably operate on an Internet-wide scale**.

3. Nowadays, for some classes of software products it is more important to achieve time-to-market than high reliability/quality. Conversely, for other software systems reliability is vital and central, as these systems need to guarantee safety critical requirements. At the same time, we have applications (e.g., mobile apps) that are used for a few weeks or even just for a specific event, while there are software systems whose lifetime spans decades, and need to be continuously evolved and integrated taking into account new waves of technologies and services available on the Internet. In general, the variety of situations and contexts in which software is used is significantly expanding. Consequently, **existing quality standards and models need to be extended and adapted to very different situations and contexts**.

4.2 Challenge 2: The Internet is the Architectural and Execution Infrastructure

Most modern systems are built by aggregating and recombining distributed software components interacting over the Internet. This notion is of course fairly obvious for web-based and mobile applications, where the Internet enables application mashups and the creation of widely distributed client-server and p2p systems. However, increasingly **any software is directly or indirectly operating or integrated over the Internet**. Even control systems and industrial applications (e.g., SCADA-based control systems) are more and more directly interacting with classical information systems, and are managed and controlled through the Internet. In general, any software element, even those strictly related to real-time and control functions, is part of wide Internet-based systems. Basically, **the classical separation among different types of software tends to disappear or to become more complex to articulate and master**.

In the past decade, (at least) three expressions have emerged to emphasize and stress this trend:

1. *Internet of things*: any product (or "thing") is becoming "intelligent" by incorporating some computing and communication capability.

2. *Smart services*: classical services interacting with "intelligent objects" become new "smart services" [1] (e.g., a connected car enables new forms of insurance and assistance services based on a pay-as-you-go approach). The boundaries of a software system span the entire value chain of a company, from traditional back ends to unconventional front ends.

3. *Disappearing computers*: computing devices tend to disappear as they are no longer uniquely associated with conventional desktop or notebook machines.

Cloud computing is another approach that reinforces the notion of **the Internet as the infrastructure for the development and operation of modern software systems**. Cloud computing is based on the virtualization over the Internet of different kinds of services and infrastructures:

Infrastructure as a Service (IaaS): provisioning of virtual machines and other infrastructural elements.

Platform as a Service (PaaS): provisioning of databases, web servers, and other runtime environments.

Software as a Service (SaaS): provisioning of software applications as remote services.

Indeed, cloud computing stresses the notion of software systems being organized as a combination of remote, distributed, and/or local components (typically, according to a "hybrid cloud" paradigm).

This trend poses new problems and challenges to Software Engineering in general, and to Software Process research more specifically. How do we support and assist developers in all the different phases of the development process? What

is the impact of the scale of the Internet on existing support technologies and methods? Are these technologies and methods able to cope with these challenges and contraints? What is the role of users and customers in this process? What are the new actors in this worldwide distributed value chain?

4.3 Challenge 3: Users are Mobile, Nomadic, and "Always On"

The technological evolution of the past decade has radically reshaped the IT market and changed users' behavior and attitude. In particular, the advent of smartphones, tablets, sensors & intelligent devices have ignited a shift in focus from classical desktop and enterprise computers to mobile devices. This change poses a number of critical and specific challenges to software developers:

- Mobile devices exhibit different form factors and interaction paradigms. Thus, user interfaces have to be completely rethought. It is not just a matter of "increasing usability": rather, it is necessary to study new interaction paradigms able to explore the intrinsic characteristics of these new devices and of the conditions under which they are typically used. In general, it is mandatory to integrate software design techniques and expertise with true industrial designers' skills and methods.
- Mobile software must operate taking into account the varying reliability of the Internet connection it may be forced to use. Thus the alignment and synchronization with back ends become crucial.
- A crucial issue for mobile devices is power consumption. This is not just a problem related to hardware components or batteries: software designers have become increasingly aware that software has to be designed in order to minimize the usage of hardware and communication resources and, consequently, reduce power consumption. Indeed, "software for low power" is now a critical area of research that is (must be!) tackled by software engineering and software process research (e.g., by providing simulation systems able to estimate and optimize the consumption of power associated with specific software architectures and design choices).

In general, **designing software for mobile devices is not just a variation of classical development processes**: it requires new and specific techniques, policies, and methods able to effectively address the new challenges introduced by this change.

4.4 Challenge 4: The Internet is the Basic Distribution and Business Infrastructure

In the past decade, software distribution and commercialization have radically changed as a consequence of two main facts: 1) devices are permanently connected to the Internet; 2) software can be easily distributed, installed, and configured over the Internet. In turn, these two changes have had important consequences on the nature and organization of the software market.

1. Software updates can be accomplished much more frequently. As soon as a bug is discovered, developers can push updates through the Internet.
2. Users and customers do expect frequent and timely updates to address faults or issues emerged by using a software application.
3. Software development is no longer constrained by national or local markets. Thus, software needs to be able to operate in multiple languages and coherently with the specific requirements and contraints of each region.
4. E-commerce has been applied also to software distribution. Of course, being software virtual "by nature", it has been very easy to replace conventional distribution channels and stores with virtual ones.

These trends have caused and generated another important innovation: *app stores*. Initially, they have been conceived to simplify the purchase of software applications on mobile devices. The first examples have been the stores developed for Windows Mobile and Symbian in the first half of the past decade. However, the advent of the iPhone App Store has introduced a **radical change with respect to the existing ones**: the App Store is the only way to legally sell and install software on an Apple mobile device. This way, the App Store introduces a new approach to software distribution: **it defines an ecosystem in which the hardware/software producer (Apple) controls the platform/OS (iOS), and the process used to distribute and sell any other software (even the one offered by others)** [22].

In the past years, other stores and associated ecosystems have been created using a paradigm similar to the Apple App Store, the most popular ones being the Mac Store (OSX), Google Play (Android), and the Windows Store (Windows). In general, app stores define a totally new software market, not only because they change the process through which software is distributed and installed, but also because they have introduced new business models (e.g., "in-app purchases") and have disrupted existing commercial approaches. The overall effect has been the advent of many new (often small) developers, an incredible growth in the number of apps being sold, and the establishment of completely different economics and revenue profiles.

In parallel with the development of app stores, the evolution of operating systems has been biased by two somewhat conflicting trends.

On one side, the role of operating systems tend to become less and less relevant:

- Many application are now web-based and therefore independent from specific operating systems (even if now they are dependent on different browsers ...).
- There are several cross-platform development environments and technologies (e.g., HTML5) that are transparent with respect to the specific operating system running on the machine/device.

On the other side, traditional operating systems and platforms have proliferated with the introduction of new families of mobile devices. In turn, the advent of mobile operating systems has influenced also the evolution of desktop and enterprise platforms. As a consequence, while in the past decades we just had Windows and a few Unix dialects, nowadays we have a number of desktop/mobile operating systems: iOS, OSX, Windows, Linux (multiple distributions),

and Android (multiple versions). In addition, there are a number of Unix versions running on enterprise architectures and solutions. Finally, even for the same platform (e.g., iOS) there might be significant differences, requirements, and constraints due to multiple sizes and form factors (e.g., iPad vs. iPhone apps).

In general, over the past years **the number of target environments that a developer has to consider in order to plan a software development initiative have increased significantly**, even if operating systems as such are less relevant than in the past. In addition, the entire software development and distribution chain has radically changed: the deployment and management of the same application across multiple stores and platforms/dialects (in particular, Android) has become extremely complex and expensive. As a consequence, **quality assurance and software security have become even more important and critical than before, as the achievements reached so far still do not allow to fully address such complexity**.

Recently, a new form of ecosystems is emerging: **open API ecosystems**. They are based on the notion of open standards, application mashup, and *coopetition*. For instance, one of such ecosystems is promoted by the US Government[16] that offers a catalog of the open APIs offered by many governmental organizations. Another example is E015, being developed for the upcoming Universal Exhibition to be held in Milano in 2015[17]. E015 enables a new approach to the design and implementation of advanced digital services and applications. In particular, E015 is a multi-stakeholder initiative that comprises a number of synergic elements:

- A digital interoperability platform (middleware) based on the Service-Oriented Architecture (SOA) paradigm, open standards, and a flat peer-to-peer interoperability model.
- A set of shared rules and policies ensuring non-discriminatory access to the ecosystem, and promoting collaboration and cooperation (coopetition) among individual developers, private companies and industries, public administrations. A developer can 1) expose its own services (API) specifying the conditions under which they can be used; 2) use the available services (according to the rules specified by the service creator) to develop its own applications.
- A governance model and related processes that ensure neutrality of the ecosystem with respect to individual needs and interests, evolution of the technical specifications, promotion and diffusion of the ecosystem across all domains and sectors of the society.
- An open participatory process that enables everybody to contribute and collaborate.

E015 is not just a single application or a physical infrastructure. E015 exploits the notion of API economy to go beyond the open data paradigm and provides full, bidirectional, and direct interoperability among autonomous distributed applications (open services). The interoperability model is based on open standards. Therefore, E015 is an open ecosystem that radically turns upside down the notion of "walled garden" used in the past decades by many telco operators.

[16] `http://www.data.gov/developers/page/developer-resources`

[17] `http://www.e015.expo2015.org`

5. RESEARCH ISSUES AND DIRECTIONS

Many of the research areas and activities identified in Section 3 are dealing with and addressing different aspects and facets of the four challenges discussed in the previous section. Certainly, the problems we as a community are facing are extremely complex and rapidly evolving. Therefore, if and when needed, it is essential to continuously reassess and redirect the focus of the research work. Obviously, this is not a simple exercise. The risk is to overlook important topics, overemphasize others, fail to spot and understand the real important ones. Still, it is possible to mention some topics that are increasingly important and still lack convincing answers.

5.1 From Rigid Compliance to Smart Convergence

Software development is a human-centered process (see Section 3.1), in which creativity and autonomy play a crucial role. Consequently, most of the activities occurring in a software process cannot be rigidly automated. Typically, there are boundaries and goals that constrain and drive software developers' work, but it is impossible to force them to follow rigid and predefined patterns and procedures. Certainly, there are specific activities where prescription and automation are useful, e.g., distributed configuration management and software deployment (see Section 3.7). Indeed, these are highly repetitive and error-prone activities and can therefore benefit from highly automated environments. But these are just a subset of the activities that defines software development.

In general, software development is a complex process in which it is not useful or possible to enforce a step-by-step compliance with a rigid predefined model. In software development consistency is the exception and inconsistency is the most common state [3, 13]. Therefore, it is impossible to impose a rigid adherence to a (often strictly) predefined set of rules and constraints. Rather, it is much more important to ensure that the process as a whole "converges" towards the desired outcome, tolerating, controlling, and exposing inconsistencies as they occur. **This "smart" convergence is much more important than rigid compliance**. In process automation and support, it is therefore essential to make inconsistencies a "first-class citizen", helping software developers to visualize, monitor, and manage them, rather than "fighting" to eradicate them.

Agile development approaches (see Section 3.2) are moving along this line, stressing the ability to address continuous changes in requirements and implement the required architectural refactoring. Unfortunately, even if the Waterfall Model is typically presented as a historical relic of ancient times and many effective Agile methods have been introduced and brought to the market, too often real development activities are still carried out according to rigid, inflexible, and fairly sequential processes. Indeed, it is essential to study new approaches, management practices, design methods, and also technical solutions to make agility and smart convergence easy to implement and "enact" in

traditional software development processes, whenever this is required or desirable.

5.2 The Fading Distinction Between Design, Development, and Operation

As discussed in Section 3.7, development and operation teams are increasingly integrated to reduce the timespan from development to operation. Moreover, as recognized several years ago [16], there is an increasingly strong connection and mutual dependency between software architectures and enabling infrastructures (operating systems, middleware, Internet-based architectures and infrastructures, cloud computing).

These observations have at least three main consequences:

1. The design of a modern software system is strictly dependent on the nature and characteristics of the infrastructure being used to implement it. It is not wise and not even possible to design a system ignoring the infrastructure that will be used to build, deploy, and manage it.

2. Even more, systems evolve as combination of classical software development activities, dynamic infrastructure evolution, on-the-fly reconfiguration of run-time components.

3. Finally, non-functional requirements related to performance management and fault tolerance are crucial and must be considered throughout the entire development process and not just as implementation issues to be dealt with "once the system goes operational".

In general, **the classical distinction among design, implementation, and operation** tends to disappear or to be radically redefined.

5.3 Process Appraisal and Visualization

One of the most important and crucial issues in mastering any complex phenomenon is being able to understand and assess its state and behavior. In Section 3.7 we have mentioned that tools like Jenkins offer metrics typically derived from the results of the execution of code analysis and testing tools. These metrics alone, however, are not necessarily sufficient to fully appraise and evaluate the state of a complex software development process, which is often distributed and sparse both geographically and from a managerial and commercial viewpoint.

In general, **it is essential to identify and develop new methods to assess, represent, and communicate the state of a software development process**. It is not just a matter of identifying new and more convincing metrics and classical measurement programs, or to conceive more appealing and easy-to-use user interfaces. We really need new paradigms (and underlying enabling technologies) that are able to intuitively convey the key information describing the state of a process, of the organization enacting it and of the software system being managed, and to support their study and assessment whenever needed or desirable.

5.4 Security, Privacy, and Trust

Software is increasingly pervasive and ubiquitous. Consequently, the issues related to security, privacy and trust become more and more critical. Security threats can emerge anywhere: management of development environments and back ends, communication between front ends and back ends, management of front ends, user interfaces, social engineering. Any phase of the value chain can be attacked and expose companies and individuals to severe risks. Moreover, the impressive development of mobile technologies and of Internet-based services, along with the rising of the debate on international monitoring activities, have made them a (if not "the") major topic also for the software engineering community.

Are software development methods and technologies adequate to assess the level of security, privacy, and trust of a software system? How do we design upfront and prove to customers and users that a software system exhibits adequate levels of security, privacy, and trust? Is this something that can and should be dealt with independently of classical software development activities? Worse, is it something that should be considered "out of scope" and not relevant for the software engineering community?

Certainly, the topic is not just technical, but, **as researchers, we can (and should) identify and assess the major issues, propose ways to monitor and manage threats, assess the impact that these problems can have on software development activities**. When should we consider them? How? To what extent? What techniques do we need? How to expose, visualize, and deal with them? As a responsible community of scientists we cannot ignore or overlook these problems and issues, as they are a major concern for citizens, enterprises, and institutions all around the world.

5.5 Controlled vs. Unplanned Interaction and Collaboration

Collaboration in software development activities goes well beyond the classical boundaries of the traditional software development company (see Section 3.3). In particular, there are some important trends to consider:

1. Cooperation among companies and between companies and their customers occur on an Internet-wide scale. Indeed, there is a significant number of tools and environments that aim at addressing this increasingly large market sector (see for instance Asana[18] and Basecamp[19]).

2. Open source forges and other Internet-based communities change the intrinsic dynamics according to which software is conceived, developed, and distributed.

3. The impressive development of cloud computing, Internet-based composition and mashup services (e.g., IFTTT[20], Zapier[21], CloudWork[22], and Mashape[23]), and, in perspective, of open API ecosystems (e.g, US Government APIs Initiative and E015) are radically transforming the notion of "collaboration" among software developers.

[18] `http://www.asana.com`
[19] `http://www.basecamp.com`
[20] `http://www.ifttt.com`
[21] `http://www.zapier.com`
[22] `http://www.cloudwork.com`
[23] `http://www.mashape.com`

Software development is occurring in new, unconventional, and unpredictable settings and scenarios. It is therefore essential to develop new approaches to collaboration and interaction that take into account these trends. It is not just a matter of supporting distributed teams as discussed in Section 3.3. It is necessary to deal with varying collaboration boundaries, dynamic and on-the fly interactions, intercompanies commercial and legal agreements. How is the software process community addressing these complex issues from a methodological and technological viewpoint?

5.6 Business and Organizational Models

The profound innovation occurring in the technical and organization approaches used to conceive and deliver software, and the introduction of mobile devices, app stores, and open source software have radically changed the organization and structure of the software market, even if in conflicting and somewhat divergent ways. On one hand, software is often considered a consumer product whose cost should reflect the trends and perceptions of that kind of market. On the other hand, software is the most critical and sophisticated component of any complex product or services. Thus, it is becoming increasingly difficult to identify business and organizational models that are able to guarantee innovation, return over investments, and customer satisfaction.

Of course, there are specific areas (real-time and control software, bespoke software for large and military organizations, embedded systems, ...) where these changes are less relevant and visible. Nonetheless, many classical software companies are forced to radically change their market positioning, the focus of their product strategies, and their core business models. Microsoft is a classical example of a company that is significantly changing its position and business model: from conventional client apps to cloud computing (IaaS, PaaS, and SaaS), from classical licensing schemes to other models including also open source. In general, the economics and business models of software are dramatically changing. It is essential to study this transformation to avoid a major risk: a general and unjustified "commoditization" of software. **Software is not a commodity, it does cost, it is not "free" as in "free beer"**.

These new and innovative business models cannot be conceived by just considering pure economic factors and contraints. They have to be defined by aligning innovation in business models with the intrinsic characteristics of innovative service and product models (e.g., how do we pay the cost of developing and using open API services?). Thus there is a role for the software engineering and software process community, as the issue cannot be just considered a matter of economists and business analysts.

5.7 "Don't flog the dead horse"

The innovations presented in Section 3 have been radically transforming the way software is designed, developed, integrated, distributed, and commercialized. In many cases they have been generated by practitioners who have derived proper solutions from their own needs. At the same time, the dramatic changes in the way we live and operate immersed in pervasive ICT technologies **are having (must have!) a profound impact on software engineering and software process research**. As researchers we should be more receptive to the approaches and tools that are emerging in the practitioners arena.

A major risk of any scientific community is to focus on the research areas that have been consolidated along the years, independently of the real impact of the results that have been achieved, and of the emerging issues that deserve new energies and focus. This induces a sort of continuous rework (often quite artificial) around the same topics and areas, even when they demonstrate to be not so important or aligned with the requests and challenges of the target domains. A nice italian expression to illustrate this notion says "pestare l'acqua nel mortaio", more or less equivalent to the English expressions "scooping water with a sieve" or "flogging the dead horse". We should be clever and wise enough to abandon our comfort zones and move to "unexplored oceans" that are certainly riskier, but undoubtedly more promising and ultimately useful. As Storey et al. discuss in their FOSE paper [27], we should learn how we can "influence the future" by producing results that are not only sound and theoretically interesting, but are also having a concrete positive impact on how people develop the software systems which the modern society is increasingly relying upon.

6. CONCLUSIONS

Over the years, the complexity and size of software have dramatically increased. Nowadays, software is a critical asset of our society, as it is the core element of any modern product, process, or service. These growing importance and relevance are posing new and tough challenges and pressure on software development companies and teams. They do need innovative and effective approaches to master the complexity of software development activities, reduce risks, ensure trust and reliability of the software solutions needed in the society. As a consequence, Software Engineering and Software Process research play an essential and central role in our scientific communities, universities, software companies, and industries in general.

Software Process research has significantly changed in the past decade. There has been a radical shift from providing modeling and automation of software development activities, to addressing a broader range of issues that take into account in a multidisciplinary and holistic way the different facets and aspects of such human-centered endeavors. This transformation demands for a continuous and neverending effort focused on studying related research domains and disciplines, combine industrial and academic expertise and knowledge, intertwine exploratory and long-term activities with empirical studies and in-field case studies.

If one word has to be used to represent this challenge, we would use the term "openness", as it is a reminder of a number of important topics and issues:

- We need to be "open" to the trends affecting the market (consumer and professional) and the society in general. Software is used in extremely different settings and is characterized by a variety of requirements and needs. There is no single approach – or even characterization of problems and issues – that can be uniquely used to drive and inspire the research work of the community.
- Software Process research is extremely complex and multifaceted. It is counterproductive and pointless not to reuse, recombine, and exploit the knowledge and expertise developed in other communities both in Computer Science and in other scientific domains.

- As researchers, we need to abandon our comfort zones and be "open" to new challenges, suggestions, ideas, and trends emerging anywhere in the society. We must be open to changing also assumptions and principles that have inspired out work over the past decades. The world has changed, the software industry has changed, we need to change accordingly. Even more, we should try to imagine and anticipate those changes in order to help shape the future.

Indeed, the work of the Software Process community is focused on a key element of Computer Science and Software Engineering research. In fact, any other branch is eventually relying on the ability of teams to construct and deliver software systems that are aligned with the requirements, constraints, and expectations of their users. This is a fantastic motivation to work in such an important discipline and, ultimately, to offer our contribution to the construction of a better society.

7. ACKNOWLEDGEMENTS

We wish to thank Gustavo Costa and all our colleagues at CEFRIEL and Politecnico di Milano for the useful discussions on the issues and challenges we present in this paper. We are also very grateful to Gerard Pompa, Margaret-Anne Storey, Damian A. Tamburri, and Alexey Zagalsky for their useful comments on a preliminary version of this paper. The work of the second author has been partially supported by the European Commission grant no. FP7-ICT-2011-8-318484 (MODAClouds).

8. REFERENCES

[1] G. Allmendinger and R. Lombreglia. Four strategies for the age of smart services. *Hardward Business Review*, pages 131–145, October 2005.

[2] V. Ambriola, R. Conradi, and A. Fuggetta. Assessing process-centered software engineering environments. *ACM Trans. Softw. Eng. Methodol.*, 6(3):283–328, July 1997.

[3] R. Balzer. Tolerating inconsistency. In L. Belady, D. R. Barstow, and K. Torii, editors, *ICSE*, pages 158–165. IEEE Computer Society / ACM Press, 1991.

[4] K. Blincoe, G. Valetto, and D. Damian. Do all task dependencies require coordination? the role of task properties in identifying critical coordination needs in software projects. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 213–223, New York, NY, USA, 2013. ACM.

[5] K. Blincoe, G. Valetto, and S. Goggins. Proximity: A measure to quantify the need for developers' coordination. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1351–1360, New York, NY, USA, 2012. ACM.

[6] L. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue. Research-based innovation: A tale of three projects in model-driven engineering. In *Model Driven Engineering Languages and Systems - 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30 - October 5, 2012. Proceedings*, volume 7590 of *Lecture Notes in Computer Science*, pages 793–809. Springer Berlin Heidelberg, 2012.

[7] G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta. How changes affect software entropy: an empirical study. *Empirical Software Engineering*, pages 1–38, 2012.

[8] M. Cataldo, J. D. Herbsleb, and K. M. Carley. Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '08, pages 2–11, New York, NY, USA, 2008. ACM.

[9] D. Chappel. What is Application Lifecyle Management? Technical report, David Chappel & Associates, http://www.davidchappell.com/whatisalm–chappell.pdf, 2008.

[10] V. Clerc. Do architectural knowledge product measures make a difference in gsd? In *ICGSE*, pages 382–387. IEEE, 2009.

[11] M. Cristal, D. Wildt, and R. Prikladnicki. Usage of scrum practices within a global company. In *Global Software Engineering, 2008. ICGSE 2008. IEEE International Conference on*, pages 222–226, 2008.

[12] G. Cugola and C. Ghezzi. Software processes: a retrospective and a path to the future. In *Software Process Improvement and Practice*, pages 101–123, 1998.

[13] G. Cugola, E. D. Nitto, A. Fuggetta, and C. Ghezzi. A framework for formalizing inconsistencies and deviations in human-centered systems. *ACM Trans. Softw. Eng. Methodol.*, 5(3):191–230, 1996.

[14] P. Debois. Devops: A software revolution in the making? *The Journal of Information Technology Management*, 24(8):3–5, August 2001.

[15] P. Deemer, G. Benefield, C. Larman, and B. Vodde. A lightweight guide to the theory and practice of scrum (version 2.0). Technical report, http://www.scrumprimer.org, 2012.

[16] E. Di Nitto and D. Rosenblum. Exploiting adls to specify architectural styles induced by middleware infrastructures. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 13–22, New York, NY, USA, 1999. ACM.

[17] M. Fowler. Continous integration. http://martinfowler.com/articles/continuousIntegration.html, 2006.

[18] A. Fuggetta. Software process: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, ICSE '00, pages 25–34, New York, NY, USA, 2000. ACM.

[19] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In L. C. Briand and A. L. Wolf, editors, *FOSE*, pages 188–198, 2007.

[20] J. Humble and J. Molesky. Why enterprises must adopt devops to enable continous delivery. *The Journal of Information Technology Management*, 24(8):6–12, August 2001.

[21] F. Lanubile, C. Ebert, R. Prikladnicki, and A. Vizcaino. Collaboration tools for global software

engineering. *Software, IEEE*, 27(2):52–55, 2010.

[22] K. Manikas and K. M. Hansen. Software ecosystems - a systematic literature review. *J. Syst. Softw.*, 86(5):1294–1306, May 2013.

[23] J. Mukerji and J. Miller. MDA Guide Version 1.0.1. omg/2003-06-01, June 2003.

[24] L. Osterweil. Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, pages 2–13, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[25] R. K. Panesar-Walawege, M. Sabetzadeh, and L. C. Briand. Supporting the verification of compliance to safety standards via model-driven engineering: Approach, tool-support and empirical validation. *Information & Software Technology*, 55(5):836–864, 2013.

[26] J. F. Smart. *Jenkins The Definitive Guide*. O'Reilly, 2011.

[27] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky. The (r)evolution of social media in software engineering. In *Proceedings of the 36th International Conference on Software Engineering, Future on Software Engineering Track*, 2014.

[28] M. Striebeck. Ssh! we are adding a process... In *Proceedings of the Conference on AGILE 2006*, AGILE '06, pages 185–193, Washington, DC, USA, 2006. IEEE Computer Society.

[29] J. Sutherland, C. Jakobsen, and K. Johnson. Scrum and cmmi level 5: The magic potion for code warriors. In *Agile Conference (AGILE), 2007*, pages 272–278, 2007.

[30] D. A. Tamburri. *Supporting Networked Software Development*. PhD thesis, Vrije University Amsterdam, 2014.

[31] D. A. Tamburri, R. De Boer, E. Di Nitto, P. Lago, and H. v. Vliet. Dynamic networked organizations for software engineering. In *Proceedings of the 2013 International Workshop on Social Software Engineering*, SSE 2013, pages 5–12, New York, NY, USA, 2013. ACM.

[32] D. Tamburrri, P. Lago, and H. Van Vliet. Organizational social structures for software engineering. *ACM Computing Suveys*, 46(1), 2014.

[33] E. Trist and K. Bamforth. Some social and psychological consequences of the longwall method of coal getting. *Human Relations*, 4:3–38, 1951.

[34] L. G. Votta and A. Porter. Experimental software engineering: A report on the state of the art. In *Proceedings of the 17th International Conference on Software Engineering*, ICSE '95, pages 277–279, New York, NY, USA, 1995. ACM.

[35] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal. Industrial adoption of model-driven engineering: Are the tools really the problem? In *Model-Driven Engineering Languages and Systems - 16th International Conference, MODELS 2013, Miami, FL, USA, September 29 - October 4, 2013. Proceedings*, volume 8107 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 2013.

[36] C. Wohlin, M. Höst, and K. Henningsson. Empirical research methods in software engineering. In R. Conradi and A. I. Wang, editors, *ESERNET*, volume 2765 of *Lecture Notes in Computer Science*, pages 7–23. Springer, 2003.

[37] W. Wu, W.-T. Tsai, and W. Li. An evaluation framework for software crowdsourcing. *Frontiers of Computer Science*, 7(5):694–709, 2013.

[38] L. Yilmaz. Modelling software processes as human-centered adaptive work systems. In P. Abrahamsson, N. Baddoo, T. Margaria, and R. Messnarz, editors, *Software Process Improvement*, volume 4764 of *Lecture Notes in Computer Science*, pages 148–159. Springer Berlin Heidelberg, 2007.