Software Development in the Post-PC Era: Towards Software Development as a Service

Sami Alajrami ✉[1], Alexander Romanovsky[1], and Barbara Gallina[2]

[1] Newcastle University, Newcastle upon Tyne, UK
{s.h.alajrami,alexendar.romanovsky}@newcastle.ac.uk
[2] Mälardalen University, Västerås, Sweden barbara.gallina@mdh.se

Abstract. Over the years, software development has evolved to meet the needs of new types of applications and to embrace new technological disruptions. Today, we witness the rise of mobility where the role of the conventional high-end PC is declining. Some refer to this era as the Post-PC era. This technological shift, powered by a key enabling technology, cloud computing, has opened new opportunities for human advancement. Consequently, the evolving landscape of software systems drives the need for new methods for conceiving them. Such methods need to: a) address the challenges and requirements of this era and b) embrace the benefits of new technological breakthroughs. In this paper, we list the characteristics of the Post-PC era from the software development perspective and describe two motivating trends of software development processes. Then, we derive a list of requirements for the future software development from the characteristics of the Post-PC era and from the motivating trends. Finally, we propose a reference architecture for cloud-based software process enactment as an enabler for Software Development as a Service. The architecture is the first step addressing the needs that we have identified.

Keywords: Software Development, Post-PC Era, Process Enactment, Clouds

1 Introduction

Software systems are playing a critical role in modern society. Many aspects of our lives (e.g transport and health care) are dependent on software. In a way, software is *smartifying* our lives through the smart X trend (phones, watches, glasses, cars, grids and cities). The list goes on leading to a *smart society* where every aspect of the society is connected to, influenced by, and dependent on software. Although, this helps addressing several societal challenges, it comes with the cost of increased software complexity. This complexity is then reflected on the way software is conceived where the expectations of quality, reliability, security, safety and fast delivery are higher than ever.

Driven by challenges and opportunities, software development will continue to evolve to address the *smart society* needs and beyond. For example, the Internet has made Global Software Engineering (GSE) possible while economical factors and market needs have motivated the rise of new development paradigms.

As Maximilien and Campos point out [10], we are entering the *Post-PC* era. This era is characterized by the increasing mobility and connectivity of people and devices, and the use of the Internet as a computing delivery medium. The role of the traditional personal computers (high-specification desktops) is gradually declining. Personal computers are becoming mobile and low-specification devices. Users can use any Internet-connected low-specification device to perform their tasks on powerful computing resources delivered over the Internet (using tools which are delivered as services). With this mobility, the relevance of OSs/platforms becomes less [7] as many software applications are offered in an OS/platform neutral fashion (e.g. services or HTML5). Cloud computing provides the enabling computing infrastructure on demand for such applications.

Accordingly, the way software is conceived needs to adapt to the rising *Post-PC* era. Software development is a complex socio-technical process which involves multiple stakeholders. Development teams use a wide range of tools/platforms for development, testing, deployment and operation of software. Some of these tools are already offered through the Internet (e.g. Eclipse Orion[3]). This paradigm is often referred to as Tools as a Service (TaaS). TaaS, however, overlooks the organizational aspects of the process. Therefore, there is a need for *Software Development as a Service (SDaaS)* which uses the cloud to support modelling, managing and enacting software processes in a model-driven fashion. SDaaS can utilize cloud as an execution and distribution medium where tools are offered as services and orchestrated in workflows. Development environments will be created on the fly and scaled as needed. Engineers will be able to do their work on-the-go from anywhere. Furthermore, modelling and monitoring the process itself will integrate the organizational and management aspects into the development environment.

In this paper, we propose a reference architecture for cloud-based software process enactment as an enabler for Software Development as a Service (SDaaS). This architecture brings the benefits of clouds and modelling to support development processes. We describe two industry-inspired development trends from the two themes: Continuous Delivery and Global Software Engineering. We highlight the impact of the *Post-PC* era on software development and identify the requirements of software development in that era. Based on these requirements, we design the proposed SDaaS architecture.

2 Motivating Trends

In this section, we list and discuss two industry-inspired motivating trends which describe different development/business needs a modern software vendor is facing. For each trend, we discuss its impact on software development.

2.1 Continuous Delivery

Continuous Delivery [9] has become a trendy software development paradigm along with DevOps. Together, they aim at bridging the gaps between devel-

[3] https://orionhub.org/

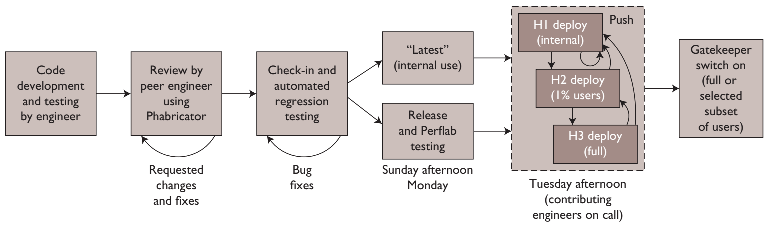
Code development and testing by engineer
Review by peer engineer using Phabricator
Requested changes and fixes
Check-in and automated regression testing
Bug fixes
"Latest" (internal use)
Release and Perflab testing
Sunday afternoon Monday
Push
H1 deploy (internal)
H2 deploy (1% users)
H3 deploy (full)
Tuesday afternoon (contributing engineers on call)
Gatekeeper switch on (full or selected subset of users)

Fig. 1. Facebook's deployment pipeline [6].

opment and operations teams and automate the build-test-deploy-release cycle. The motivation is to achieve frequent releases, reduce conflicts and therefore, reduce cost. To achieve such automation, teams should follow certain practices and use supporting tools/platforms. Humble and Farley [9] set the principles and technical practices for successful implementation of Continuous Delivery. We use Facebook's deployment pipeline [6] as an example of a Continuous Delivery process for large projects. Facebook is an example of a complex software that requires rapid innovation and release of new features. As shown in Fig. 1, the release cycle for each new feature starts by engineers coding a new feature or a bug fix. The code is then reviewed by a different engineer using the *Phabricator* code review tool. Tools such as distributed source control and automated testing packages are used. The code is released on stages: first it is released to internal employees to test it and is also tested for performance issues using *Preflab*. Then (after fixing any discovered issues), it is released to a small portion of users using the *Gatekeeper* tool. Only after these stages have passed successfully, the new feature would be released to all users.

Discussion Facebook is delivered through the Internet and changes and new features are continuously pushed to users transparently. This means that developers will be committing and integrating code very often (sometimes on daily basis). The benefits of such frequency includes maintaining a bug-free code base and easier bug fixing (since searching for bugs is limited to last pushed code). Automation and repeatability of the software build-test-deployment-release are a key enabling factor to Continuous Delivery. To pick up the fruits of Continuous Delivery, the social/organizational aspect must be considered. For example, if developers do not commit their code regularly, the Continuous Delivery chain is broken. Therefore, there is a need for convergence and monitoring support to ensure certain processes and practices are followed.

2.2 Software Outsourcing

The *Post-PC* era is also a globalized era. Software development outsourcing was driven by business and economic factors (e.g. exploiting low-cost developers and reducing the time-to-market). In addition, companies tend to outsource the tasks that they lack the skills or expertise to perform. Outsourcing can take place either within the same organization (intra-organization) or across organizations (inter-organization).

This example is inspired by the railway system development. In this scenario, there are two companies cooperating on system development. Company A is a contractor that runs large industrial projects for designing/redesigning railway networks. Among various tools the company uses a number of simulation tools to visualise and analyse the systems it is building, to debug them, to check their characteristics (such as throughput, energy consumption, performance and capacity). During such projects company A develops a wide range of models, diagrams, documents and blueprints that will be used for building the network. As part of this work, company A needs to develop a safe signalling software to operate the network by following a stringent software process. To ensure the system safety, company A would like to use industry-strength formal technologies. Company A does not have expertise in conducting large-scale formal verification of complex systems so it decides to outsource this work to small independent company B that has the right skill set. Conducting this type of verification is the main business of company B. The artefacts to be used by company B include layouts, infrastructure data, service patterns, timetables and control tables. Due to the confidential nature of these artefacts, company B signs a non disclosure agreement and a Service Level Agreement (SLA) with company A and as a precaution, it undertakes all its processes on a private infrastructure. Both companies (A and B) only exchange relevant artefacts and do not know each other's internal processes.

Discussion In reality, large scale projects may include intra and inter-organization outsourcing with other teams/partners. Management of such projects can be tedious and consumes enormous resources (time and effort) to monitor and synchronize the different outsourced sub-projects. Several issues may arise. Small issues such as using different tool versions by different teams may easily go unnoticed till a late stage of the project at which it will become very costly to fix. Other concerns include how to ensure the quality of the outsourced tasks and how to monitor that they have been performed according to SLAs. Process-state-awareness and communication is vital for the success of such distributed development projects [7]. Therefore, there is a need for efficient management and monitoring of such projects.

3 Characteristics of the *Post-PC* Era

The term *Post-PC* era was used to describe the fall of PC sales due to the rise of mobile devices. When David Clark used the term for the first time in a talk called "The *Post-PC* Internet" in 1999, he predicted that the future will be "inevitably heterogeneous" and "a network full of services" [4]. Today, we can see this

[4] http://www.nytimes.com/1999/04/18/business/economic-view-is-mr-gates-pouring-fuel-on-his-rivals-fire.html

prophecy taking place in the form of heterogeneous mobile devices and services while PCs are becoming more portable and low-specification. The technology shift in this era is enabled by cloud computing technology and the Internet. This shift has changed the way users access and interact with technology. We categorize the characteristics of the *Post-PC* era into two categories: a) technical and b) organizational:

3.1 Technical Characteristics

The Rise of Mobility. Over the past few years, mobile devices have been shaking the dominance of PCs. Users use mobile devices for many daily activities. This has enabled new business models and new software distribution platforms (e.g. app stores) [7]. Consequently, users have become more mobile and have adopted new interaction patterns for interacting with technology (e.g. touch and voice). This increasing mobility impacts software development in two ways: one impacts the produced mobile software (e.g. to have less power consumption) and the other impacts the development process itself. The new interaction paradigms that came with mobile devices have driven new works on unconventional development methods. Microsoft *TouchDevelop* [4] platform enables programming on the go using only mobile phone touch screens. Another trend is using voice recognition to input code [5]. **The Cloud as the Development and Operation Platform.** Mobile devices have limited computing power. To overcome this challenge, mobile applications delegate the processing and storage to cloud platforms over the Internet. Cloud computing allows acquiring computing resources on the fly and on a pay-as-you-go pricing model. This paradigm has enabled Software, Platform (hardware, OS, etc.) and Infrastructure to be offered as services over the Internet. Consequently, software development is increasingly relying on Internet services which enable collaboration and integration between development teams (e.g. Github [6]). Open source software and crowdsourcing are examples of how the Internet (powered by the cloud) enables collaborative development. In addition, many software systems are now built by aggregating other services from the Internet. Cloud is becoming the development and the operation environment for software. This trend raises the need for alternative methods and technologies to conceive, design, implement, test, deploy and evolve software [7].

3.2 Organizational (Business) Characteristics

On Demand Infrastructure and Tools Acquisition. With cloud and services, traditional software distribution models have changed. Desktop clients are being changed to cloud-based tools and mobile applications. Computing infrastructure is now only acquired and scaled up/down as needed. Along with this shift, pricing models have also changed from the desktop client licence model to in-app purchases and pay-as-you-go models. **Globalized Development.** As

[5] https://www.youtube.com/watch?v=8SkdfdXWYaI
[6] https://github.com/

mentioned earlier, the *Post-PC* era is driving the development and operation to take place in the cloud. This has facilitated undertaking global software development projects. Software development outsourcing helps reducing costs and development time, but also introduces management challenges to overcome spatial, cultural and geographical distances in order to ensure the quality of the product and effective communication between development teams. **Dissolving Boundaries.** The Internet has made geographical boundaries within or between companies disappear. In addition, team boundaries are also fading [7]. Design, development, testing and operations are no longer isolated tasks. Trendy development paradigms such as DevOps calls for tight collaboration and integration across these tasks.

4 Software Development in the *Post-PC* Era

The characteristics listed in the previous section affect how software development is going to be conducted in the near future and raises the need for new methods and tools for software development. Here, we list a non-exhaustive list of requirements (derived from Sections 2 and 3) for the next software development environments: **Process Monitoring & Management.** Regardless of which process model you use, the need for process visualization, monitoring the process status and detecting/predicting problems and deviations becomes vital. Considering the outsourcing scenario in Section 2, visual models of the process would ease communication and understanding between distributed teams. Process monitoring and status checking would help project managers to identify bottlenecks in the process; **Tools as a Service.** The process models contain the tools needed to support the process. To achieve executability of models, the required tools should be available as a service over the Internet. While some tools can be automated, others can be interactive. Interactive tools should provide interaction patterns over the Internet. Consistency of tool versions used by distributed teams for development and production is vital. As Humble and Farley [9] demonstrate (using their experience from real-world projects), using different versions of the same package by collaborating teams could create very costly problems; **Provenance, Governance & SLA monitoring.** Software development is a human-centric process and when the involved humans are distributed within the same or across different companies, effective management becomes essential. As mentioned earlier, process monitoring and consistency checks are important, but they are not enough. Data about the process, its enactment environment, the tools used, the stakeholders involved and the artefacts produced/consumed should be logged. Such data can be useful for process improvement and accountability. Moreover, when multiple companies are involved in a project, the processes followed by both parties should comply with the agreed SLA. Therefore, there is a need for SLA monitoring to assist the management of such collaborative projects and ensure all parties are compliant; **Artefacts Management.** Artefacts are tightly related to the previous needs and process models are artefacts themselves. Therefore, artefacts should be managed and stored effectively. They should be accessible from anywhere and available at any time. Changes made to them should be tracked and different versions of an evolving artefact should be kept; **Automation.** The question about how much automation one can have in a software process is important. The answer is indeed, a limited portion. However, automation when possible is beneficial. Repetitive tasks such as the build-test-deploy-release cycle are error-prone and their automation can prevent errors and save time. Non-interactive tasks (e.g. testing or model checking) can be automated. Furthermore, automated background service can be run to check consistency and compliance and monitor SLAs.

5 Reference Architecture for Enabling SDaaS

Aggregating the previous needs leads to Software Development as a Service (SDaaS). SDaaS provides tools for modelling, enacting and managing software processes. It enables orchestrating tools on the fly as services and manage and store artefacts in the cloud. In addition, it enables utilizing the scalable cloud resources to run automated processes and meet the needs of computing-intensive tasks (e.g. code analysis and testing). In this section, we propose a reference architecture for SDaaS. The architecture is model-driven where processes are modelled and enacted as workflows.

The architecture complies with the Workflow Management Coalition (WfMC) reference model [8] and is designed as a service. It consists of three main components: a) **The modelling and management** interface is offered as Software as a Service (SaaS) and allows distributed teams to access, model, enact and manage processes. b) **The enactment service** is offered as a Platform as a Service (PaaS) and handles the instantiation, enactment and monitoring of process models. And c) **Workflow Engines** are deployed in a set of hybrid clouds and enact the individual workflow tasks/activities.

5.1 Process Modelling (Build Time)

Software processes consist of a set of different types of (e.g. interactive or automated) activities, which are to be enacted by different stakeholders with different enactment requirements (e.g. privacy, computing power). These process details need to be captured. Software & Systems Process Engineering Meta-model (SPEM2.0) [11] is the Object Management Group (OMG) standard for modelling software processes. SPEM2.0 lacks explicit support for expressing cloud-based process enactment and control flow semantics. Consequently, we proposed EXE-SPEM [2] which is an extension of SPEM2.0 for cloud-based enactment. Software process models modelled in EXE-SPEM can be mapped to an executable XML notation.

Fig. 2 shows the software process build time components which are packed as a SaaS solution. The **Model Authoring** module allows constructing process models using EXE-SPEM constructs. The **Access & Sync. Service** applies access management policies and ensures the consistency of models that

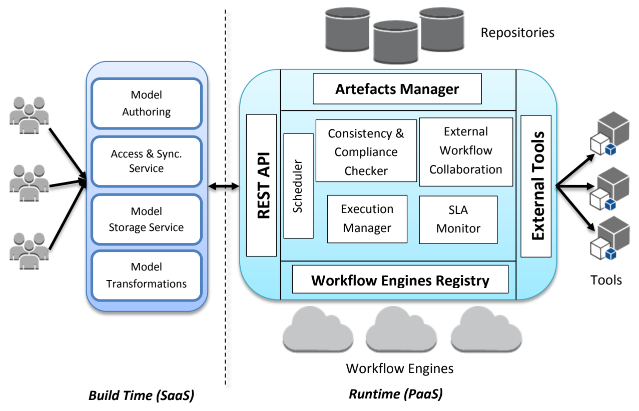

Fig. 2. Detailed architecture for the Software Development as a Service (SDaaS) platform.

are being authored by distributed teams simultaneously. This module also notifies collaborators when a model is changed/updated. Once the model is authored, the **Model Storage Service** allows saving/retrieving the model into the cloud-based repository through the enactment service API. Finally, models can be transformed into the executable XML notation from EXE-SPEM using the **Model Transformations** module.

5.2 The Enactment Service (Runtime)

The enactment service has an API to interact with the process modelling service. This way, modelling can be done from SaaS or a plug-in for a legacy desktop client. Behind the API, the service is responsible for the runtime instantiation and execution of process models. To do this, the service consists of several modules as illustrated in Fig. 2. These modules are: **The REST API** provides endpoints for process enactment and monitoring and artefacts storage and retrieval; **The Artefacts Manager** stores the artefacts and meta-data about them into the artefacts repository. Software processes involve producing large number of artefacts such as: code, models and documentation. These artefacts capture invaluable information about both the software process and product evolution. The artefact meta-data includes: actors involved, version, tools used and the date and time the artefact was created/modified; **The External Tools** are service blocks performing the process activities. These blocks are either: interactive, control points (providing control flow during the process execution) or automated fire-and-forget activities. This module provides the necessary information on these activities when needed for process execution; **The Execution Manager** orchestrates the enactment of process models. First, an instance of the model is created and the ready-to-execute activities are passed to the scheduler. The scheduled activities are then executed on workflow engines. During the execution of the process, the execution manager tracks of the status of the process instance being executed. This module also logs all the provenance data about each process instance execution; **The Workflow Engines Registry** is responsible for starting, stopping and monitoring workflow engines based on the activities scheduling policies used by the scheduler. Workflow engines are independent applications running on different cloud providers. Activities get executed in a workflow engine that is deployed on a public or private cloud. The workflow engine has to meet the execution requirements expressed in the process model. The execution of activities is a black-box execution which means that the workflow engine would not know any information about the process being executed. This reduces the risks of privacy and confidentiality breaches. In order to decouple the enactment service from the workflow engines, asynchronous communication between them is achieved through message oriented middleware; **The Scheduler** handles the planning of process execution. This involves checking the needed resources (from the process model). The scheduler should operate using a policy to meet the the enactment requirements (e.g. enacting an activity on a private cloud) while minimizing the cost. Several cloud-based workflow scheduling algorithms exist and can be used (e.g. [1]). The schedules generated by the scheduler determine the expected load of execution and is used by the workflow engines registry to dynamically scale the number of workflow engines; **The Consistency Checker** automatically checks the process consistency during its execution which can alleviate problems early and save time and cost (as explained in Section 4). Discussion of consistency checking techniques is beyond the scope of this paper; **The SLA monitor** transparently ensures that all parties collaborating on a project are not breaching the agreed SLA (as explained in the software outsourcing scenario in Section 2). While each organization can have its own SDaaS environment, these environments can exchange data about the process state and execution using the External Workflow Collaboration module; Finally, **The External Workflow Collaboration** allows process execution to incorporate invoking processes managed by another workflow system (e.g. from a different company).

6 Conclusion

The *Post-PC* era is here and software is embedded in almost every aspect of our daily life. Software systems have evolved but the way they are conceived still needs to be rethought to adapt to the new era's challenges and to embrace its technological breakthroughs.

In this paper, we have described the characteristics of the new era and its impact on software development. We also proposed the SDaaS reference architecture for supporting software processes enactment. To become a reality, this proposal requires tools to be offered as services. We have developed a prototype of the proposed architecture consisting of an enactment engine that executes software processes, a number of off-the-shelf tools deployed as services in our tool repository and an artefact store. The prototype was used to enact a safety-related process [3] and a number of verification/modelling processes. Our ongoing work focuses on implementing larger and more complex processes and evaluating the architecture proposed. In a longer run we aim at creating a community of developers extending the architecture and applying it for the development of complex software systems.

Additionally, Empirical studies are needed to study the effects of this proposal on the organizational, technical and economical aspects of software development processes. Furthermore, the effect on different development process models (e.g. Agile) also needs to be analysed and benchmarked. Usability studies can determine the effects this approach may have on individual developers, managers and other stakeholders. Indeed, as Fred Brooks put it, *"There is no silver bullet"* and we can only eliminate accidental difficulties in software development. Inherent difficulties will continue to exist as software and its development evolve [5].

References

1. Abramson, D., Lees, M., Krzhizhanovskaya, V., Dongarra, J., Sloot, P.M., Wang, J., Korambath, P., Altintas, I., Davis, J., Crawl, D.: Workflow as a service in the cloud: Architecture and scheduling algorithms. Procedia Computer Science 29, 546 – 556 (2014)
2. Alajrami, S., Gallina, B., Romanovsky, A.: Exe-spem: Towards cloud-based executable software process models. In: Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development, MODELWARD '16. pp. 517–526 (2016)
3. Alajrami, S., Gallina, B., Sljivo, I., Romanovsky, A., Isberg, P.: Towards cloud-based enactment of safety-related processes. In: Proceedings of the 35th International Conference on Computer Safety, Reliability and Security, SafeComp '16. vol. LNCS 9922 (2016), to appear
4. Ball, T., Burckhardt, S., de Halleux, J., Moskal, M., Tillmann, N.: Beyond open source: The touchdevelop cloud-based integrated development environment. Tech. Rep. MSR-TR-2014-127, Microsoft Research (September 2014)
5. Brooks, F.P.: No silver bullet: Essence and accidents of software engineering. IEEE Computer 20, 10–19 (1987)
6. Feitelson, D., Frachtenberg, E., Beck, K.: Development and deployment at facebook. IEEE Internet Computing 17(4), 8–17 (2013)
7. Fuggetta, A., Di Nitto, E.: Software process. In: Proceedings of the on Future of Software Engineering. pp. 1–12. FOSE, ACM (2014)
8. Hollingsworth, D.: Workflow Reference Model. No. TC00-1003, Workflow Management Coalition (WfMC) (January 1995)
9. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation. Addison-Wesley Professional, 1st edn. (2010)
10. Maximilien, E.M., Campos, P.: Facts, trends and challenges in modern software development. Int. J. Agil. Extrem. Softw. Dev. 1(1), 1–5 (Jul 2012)
11. OMG: Software and Systems Process Engineering Meta-Model Specification, V2.0 (April 2008)