Challenges from the Cluster on Software Engineering for Services and Applications

This document summarizes the challenges that we have identified as medium-term issues to be addressed in the next 2017-2018 workprogramme, in the area of software engineering. In particular, these have been distilled from the discussions and a survey we run within the cluster as well as from the panel we have organized at ICT 2015.

Process, methodologies and productivity

Software process is a well investigated research area, but nowadays there are several new advancements in technology and practice that introduce significant changes in this aspect.

There should be a new notion of productivity: While before the development team did not have many external resources available, now personal investigation can be replaced with posting problems on well-known repositories such as stack overflow or looking at what others answered to address similar problems. Within this context, "lines of code" is not anymore the right measure of productivity. Software should be measured in terms of its other qualities, usability, reliability, scalability, …

In addition, the *new possibilities to easily gather user feedback and monitoring information* have the potential to *enable an informed evolution of software*.

The development cycle has been shortened significantly. This is due to the needs of the market and it is enabled by the availability of automation techniques for testing and deployment. However, today only proficient developers can really tackle this challenge and exploit the available technological support. We need to *improve software production methodologies* (incremental and iterative design/development approaches, verification and validation, bug finding and fixing, …) *to actually enable controlled management of such short development cycles.*

More and more companies are building software to offer it as a service. Thus, software engineering should take into account the interplay between development and operation. *DevOps is a movement focusing on understanding this interplay: how to customize it on specific organizations? How to help organizations adopting it?*

Within the context of a DevOps approach it is advisable to *shift deployment decisions and resource management from the deployment phase to the design phase* of software engineering. This allows the design to ensure that the software can make efficient use of resources. Approaches to support architecture level analysis and optimization of deployment decisions are still preliminary and should be re-enforced.

The new technological frameworks require a deep knowledge of their internals. However, it could be *counter-productive if not impossible for small companies to hire specialists* for these technological frameworks. *Can we delegate more to some methodology/platform that supports the development?*

The basic approach in software industry is still divide and conquer. Is it still the right one? Does it work in a distributed environment where we have emerging properties?

Application contexts

Software engineering is already lagging behind in the development of models, methods and design tools for IoT/CPS-enabled applications and to support the emerging trend of fog computing, where applications will be able to use and access a richer spectrum of devices and physical objects. The lifecycle of these applications is still unclear and the research community will need to develop appropriate modelling abstractions and quality-of-service analysis tools for such applications.

In these contexts, *software-aware of hardware* could take advantages of the potentials of the second and execute in a more efficient way.

Another application context that requires specific attention is big data. While technologies and middleware for big data applications have been developed, proper design, runtime management and quality assurance tools are not available yet. *The goal should be to simplify the adoption of the various complex big data technologies*.

Design patterns and systems of systems

Is it possible to design software efficiently and to enforce system of system reliability? It is not just a matter of the software under development but of the environment in which the software is developed and of the one in which it operates. Disclaimers and proclaimers associated to the software, stating, for instance, "this software is secure only if the container does this" could be very useful.

The community should make an effort to develop patters at the architectural level that describe also the obligations/constraints to be fulfilled by the system in which the software is running, and to validate and standardize them, at least within a certain application context. Also, it should face the problem of how to apply such patterns into a dynamic environment where the application context changes continuously.

Quality guarantees

We talk about software engineering since 1968. However, *the approach to coding today is wrong because it requires significant bug fixing*. We should think at a development approach that, by construction, limits the bugs of our systems.

Techniques for *assessing and optimizing QoS are emerging but, at the same time, they seem to apply only to small cases*. How to make them scale to larger contexts?

How to support the development of *catalog of well-defined software quality benchmarks*?

Requirement engineering

We still struggle in understanding what users like or not. *There is still a huge gap of competence/methodology to turn these ideas into good software*. Software is so intangible. We need more research to be able to really grasp complexity of user requirements. On the one hand, customer wishes are not always translatable into something feasible. On the other hand, technical people are not always able to properly understand requirements and express them. A model-driven approach cannot solve the translation problems from requirements to their formalization. The traditional decomposition approach can be adopted after requirements are understood.

There is the need for a complete different approach to capture emerging behavior and requirements.

Privacy and security

Privacy, big data and complex applications: How does privacy fit in information systems handling big data? Which code at which boundary is responsible for some privacy leak?

Secure computation: issues such as data structures for secure computation, approaches for establishing optimality of the level of encryption to use, theory of responsibility, accountability are still completely open.

Maintainability

Software updates need new ways to be applied. We also need self-audit for maintainability

Big Data for software engineering

Big data are useful in software engineering for:

- Studying evolution / discontinuation of application frameworks, open source components, etc.
- Analysing user trends and preferences
- Identifying feature and performance improvement opportunities
- Identifying root causes of software failures based on massive log files (>> GB)
- At runtime, gather insights on symptoms and context changes to trigger the right adaptations and perform predictive and prescriptive analytics for proactive planning and preparation of adaptation actions

As such, *software engineering should understand how to collect/filter/manage/use such big data to support the development and operation process*.