Networking session on Software Engineering and Technologies for Services and Applications
21/10/2015 at ICT2015

Contributions from the discussion

Process, methodologies and productivity
There should be a new notion of productivity: While before you needed to address all problems yourself, now you do not investigate yourself, you post your problem on stack overflow or look at what others answered to address similar problems.
So, lines of code is not anymore the right measure. You should see how your software is usable, how it is reliable, scalable, …
The methodologies to develop software should be more incremental.

The new technological frameworks require a deep knowledge of their internals. Is the software engineer who should be able to build high quality software or is it up to some methodology/platform that supports the development? Isn't counter-productive if you need to have a specialist for these technological frameworks?

Design patterns and systems of systems
Is it possible to design software efficiently and that enforces system of system reliability? Design patterns could be useful in this context. I do not mean programming patterns, but patters at the architectural level that describe also the obbligations/constraints to be fulfilled by the system in which the software is running. In fact, we have a system of systems problems. It is not just a matter of the software under development but of the environment in which it is developed. 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 design patters and to validate and standardize them, at least within a certain application context.

The point above assumes that you are living in a stable environment. Today instead we are considering a very dynamic environment. Can the approach you have in mind be applicable also in a dynamic context?

Quality guarantees
We talk about engineering since 1968. What tool do we need to prove the system does not break down?
The approach to coding today is wrong because it requires a lot of bug fixing. We should think at a development approach that, by construction, limits the bugs of our systems.

Requirement engineering
We still struggle in understanding what users like or not. Still huge gap of competence/methodology to turn these ideas into good software. Software is so intangible. There is no enough support to really grasp complexity of user requirements.

The problem is when requirements are expressed by technical people. Involve more people to define requirements that are then sent to software engineering. DevOps need to become reality.

Involvement of end users should also occur during Ops. Today we have a huge opportunity as we are able to track users and their feedback.

A model-driven approach cannot solve the translation problems from requirements to source code. You are still using the traditional decomposition approach. There is the need for a complete different approach to capture emerging behavior.

Customer wishes are not always translatable into something feasible.

Divide and conquer
The basic approach in software industry is still divide and conquer. Does this work in a distributed environment where we have emerging properties?

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

Secure computation: there is no enough work on data structures, no established optimality about the level of encryption to use, no theory of responsibility, no accountability

Maintainability
Software updates: they need new ways to be applied.
We need self-audit for maintainability

Big Data for software engineering (this has not explicitly discussed during the session but it is a contribution I got right after it)
Big data are useful 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