Foreword

The ability of handling change has been an innate property of software systems. Although very brittle when dealing with some flaws, overall software has proven to be, within certain limits, sufficiently malleable towards human imperfections (related to the specification, implementation, evaluation and operation of systems). The reason for this is that software engineers have successfully dealt with a wide range of changes, mostly at development-time, by defining appropriate processes, techniques and tools target to emerging needs. This has come at a cost by developing systems that are not able to account for variations that might be known during development-time. A source for these variations is the incorporation of changes that are expected to affect the system after its deployment. As a consequence, any changes that emerge at run-time that are not considered at development-time, need to be handled at the next maintenance cycle. This has proven to be a costly way of developing systems that is not sustainable at the long run since the software landscape is ever evolving, with increasing levels of system complexity regarding both system behaviour and structure.

New approaches are needed for handling changes since these cannot be dealt exclusively at development-time. For example, not all changes can be foreseen at development-time when disparate components (or component systems) interact, which may lead to emergent behaviours. Changes also need to be dealt at deployment- and run-time, without interrupting the services provided by the system, and without any human involvement. This is particularly the case of variability-intensive systems. These are systems that can be derived from a single specification, and because of that they can be easily modified in order to handle change. Hence the demand to define new development processes that are able to produce, deploy, operate and maintain software that is effective, efficient and provable.

When dealing with variability-intensive systems, amongst the several promising approaches, dynamic software product lines (DSPL) and self-adaptation (based on an explicit feedback control loop) might provide the appropriate foundation for architecting resilient systems (i.e., systems that support the persistence of service delivery that can justifiably be trusted, when facing changes). Both DSPL and self-adaptation are able to deal with change at run-time, and architectures take a centre stage when reacting to change (at least for some classes of self-adaptive systems).

These two approaches should not be seen as competing in their usage, they are complementary. Moreover, complementarity is not restricted to which circumstances one or the other approach should be used, complementarity is also related to their combined usage. In other words, a feedback control loop (the basis of self-adaptive systems) should be able to manage effectively and efficiently the processes associated with dynamic software product lines. How this can be achieved, it is not clear if we consider the whole process starting from feature modelling.

In the context of resilience, depending on the criticality of the system, different degrees of assurances are required, and here dynamic software product lines might be useful in assisting the structuring of evidence. The latter is fundamental when building arguments

that justify the level of trust that can be placed on the system - the 'provable' factor mentioned above. Based on this, we identify two key processes that should be related at system deployment-time: the process responsible for the provision of services and their quality, and the process responsible for the generation, collection and analysis of evidence to be used in the formulation of assurance arguments. These processes are related because of the decision whether to deploy or not a modified/adapted system. The formulation of assurance arguments, and their evolution as the system adapts, should also be supported by a dynamic process since small system variances may require a different line of assurance argumentation. Again, this is not the case for “one size fits all” since variations are expected between arguments. That is the reason why the motivation behind software product lines should also be inspirational when defining new approaches for formulating assurance arguments when building, deploying and operating resilient variability-intensive systems.

Rogério de Lemos
April 2018