Performance Requirements and Agile Projects

Alex Podelko

It looks like agile methodologies are somewhat struggling with performance requirements (and non-functional requirements in general). Probably there are several reasons for that. One is that actually even traditional software development methodologies and processes never came with a good approach to handle performance requirements. They are, of course, considered in both literature and practical projects – but are usually handled rather in ad hoc manner. Actually the process of gathering and elaboration of performance requirements is rather agile in itself, and attempts to make it rigorous and formal look unnatural and have never fully succeeded – so it should be easier and more natural to do it as part of agile methods. Still the challenge of handling multidimensional and difficult to formalize performance requirements remains intact and the difference is rather in minor adjustments to agile processes than in the essence of performance requirements.

Another reason is that practical agile development is struggling with performance in general. Theoretically it should be a piece of cake: with every iteration you have a working system and know exactly where you stand with the system's performance. You shouldn't wait until the end of the waterfall process to figure out where you are – after every iteration you can track your performance against requirements and see the progress (making adjustments on what is already implemented and what is not yet). Clearly it is supposed to make the whole performance engineering process much more straightforward.

Unfortunately, it looks like it doesn't always work this way in practice. So such notions as "hardening iterations" and "technical debt" get introduced. Although it is probably the same old problem: functionality gets priority over performance (which is somewhat explainable: you need first some functionality before you can talk about its performance). So performance related activities slip toward the end of the project, and the chance is missed to implement a proper performance engineering process built around performance requirements.

Another issue here is that agile methods are oriented toward breaking projects into small tasks, which is quite difficult to do with performance (and many other non-functional requirements) – performance-related activities usually span the whole project.

There is no standard approach to specifying performance requirements in agile methods. Mostly it is suggested to present them as user stories or as constraints. And the difference is not so much in the way the requirements are presented, both ways rather use plain text.

User stories assume using a user voice form. Cohn, for example, suggests using the "As a <type of user>, I want <some goal>, so that <some reason>" template for user stories (although he cautions that the user story template should only be used as a thinking tool, it should not be used as a fixed template). For constraints, both traditional expressions and user voice forms may be used.

The difference between user stories and constraints approaches is not in performance requirements per se, but how to address them during the development process. The point of the constraint approach is that user stories should represent finite manageable tasks, while performance-related activities can't be handled as such because they usually span multiple components and iterations. Those who suggest to use user stories address that concern in another way – for example, separating cost of initial compliance and cost of ongoing compliance.