Software Development: Product or Service?

10 June 2016 | **Software Engineering** | **Mike Hogg**
Reading time: 4 minutes

The distinction between a product and a service is clear in many cases. A car part supplier selling a battery is providing a product; a chauffeur is providing a service. The categorisation is often indicated by the commercial model – the total price of the battery can be agreed before delivery, while a chauffeur will be paid based on the time worked. When a software development team is engaged to deliver a software solution that might suggest a product categorisation; the team is building a product so surely they should deliver at a fixed price?

Consider a car repair mechanic. A mechanic may operate in the product space if they agree an upfront price for their work. They may be happy to do so where the full scope is well defined, such as changing a tyre. An issue such as a fuel pressure warning light is a different proposition however; the problem could be with the injector jets, fuel pump, pressure sensor or something else. A mechanic would not offer a fixed price on work where exploratory work is required and the solution has yet to be determined. The majority of a mechanics' work is in this latter service category – indeed simple clearly defined work like tyre changing has moved to cheaper single skill suppliers.

When off-the-shelf software provides an acceptable solution then a product model is appropriate. Similarly there are cases when the bespoke software development required is well defined; for example porting an application to a later version of an OS. However most of what we do is closer to the mechanic's service model. Our specifications rarely if ever capture the full detail of what is needed; we need to do exploratory work to better understand the problem before the solution can be defined. Similarly there are technical risks which may or may not create issues to be addressed.

Pressure a mechanic to provide a fixed price for the warning light issue above and either the price will be high to reflect the worst-case scenario and/or they'll have to cut corners in some way, such as only addressing the most probable cause and hoping this helps. This compares well to our software delivery experience. Fixed price arrangements lead to a higher overall cost as they must take a pessimistic view of the risks, and they force assumptions to be made up front which often leads to a suboptimal solution.

If committing a mechanic to a fixed price repair is not standard practice, how do we make sure that we are not paying more than we need? First and foremost is trust, ideally built from having been pleased with previous work and knowing that the provider is motivated to impress and win future work. Trust allows us to engage the mechanic based on their estimate

of what is needed. When trust has not been established then references, recommendations and qualifications are essential, and its advisable to start with smaller pieces of work. Software development is little different in these respects.

Established practice for mechanics is to complete exploratory work to identify the problem's cause, share the findings with the client and subsequently agree further work before proceeding. Where the solution cannot be fully understood (fuelling issues like that noted above are very hard to diagnose) then parts are replaced step by step until a solution is reached, with the client consulted at each step. This has parallels to software projects that start with an Inception phase where the key technical risks are addressed via prototyping and other exploratory work is performed, resulting in a revised estimate for the project as a whole. Delivery of software incrementally via iterations mirrors the step by step approach, confirming as we proceed that the client approves of the work done and agrees with the next steps. Like the mechanic we can agree a much firmer price as we embark on each iteration.

That's enough about car repair. The analogy only goes so far because in software development we are not just fixing, we are creating new features, new user interfaces, new concepts. This is a significantly higher level of complexity. We try hard to describe what is required up front, but as the solution develops we discover more about what the client really needs, and so the requirement evolves. We should consider ourselves to be service providers. We should focus on approaches that give our clients a high degree of oversight and control over what our service is delivering. We should seek to maximise the value provided by our service rather than squeezing ourselves in to a product world.