The Sweet Spot of Engineering Quality

29 July 2019 | **Product Engineering** | **Daniel Mölle**, **Jens Marggrander**
Reading time: 9 minutes

If someone assembled a list of the most alarming terms for project managers and project sponsors, then "Overengineering" and "Gold Plating" would most likely rank top. In this post, we will explore causes and countermeasures for these expensive mistakes – and discuss a high-level guideline for how much engineering and how much physical quality is just right.

Facets of Quality

The title of this post may sound confusing to people who have never been involved in product development. After all, quality is a good thing, so how can the sweet spot be anything else than "maximum quality"? Let us shed some light on the matter by investigating what quality means.

The first thing to note is that the quality of a product is a multi-dimensional property: To use the terminology of standards like ISO 25010, there are many different quality characteristics. Examples from that standard include functional suitability, usability, reliability, security and maintainability. Other examples are the safety (called "freedom from risk" in ISO 25010) and the producibility (not covered by ISO 25010 since it is a software standard) of a system. The latter, for example, describes the degree to which a product is easy to manufacture, which includes aspects such as ease of assembly or readiness for end-of-line testing.

The second important fact is that achieving high ratings in a quality characteristic comes at a price: It will affect the development effort and/or the cost of goods – and thus the final price of the product to be paid by the consumer. Hence, it is not automatically desirable to strive for maximum quality in all dimensions. Instead, there will always be a prioritization in real-world scenarios: Some quality characteristics need to be fulfilled to a larger degree than others.

The third crucial aspect is that quality characteristics are not entirely independent or positively related (now that would be nice), but conflicting. For instance, if you want to achieve incredibly large degrees of security, you will usually have to pay a high price in usability – an experience you probably know from everyday life.

Ten most common passwords 2018
(see: Wikipedia)

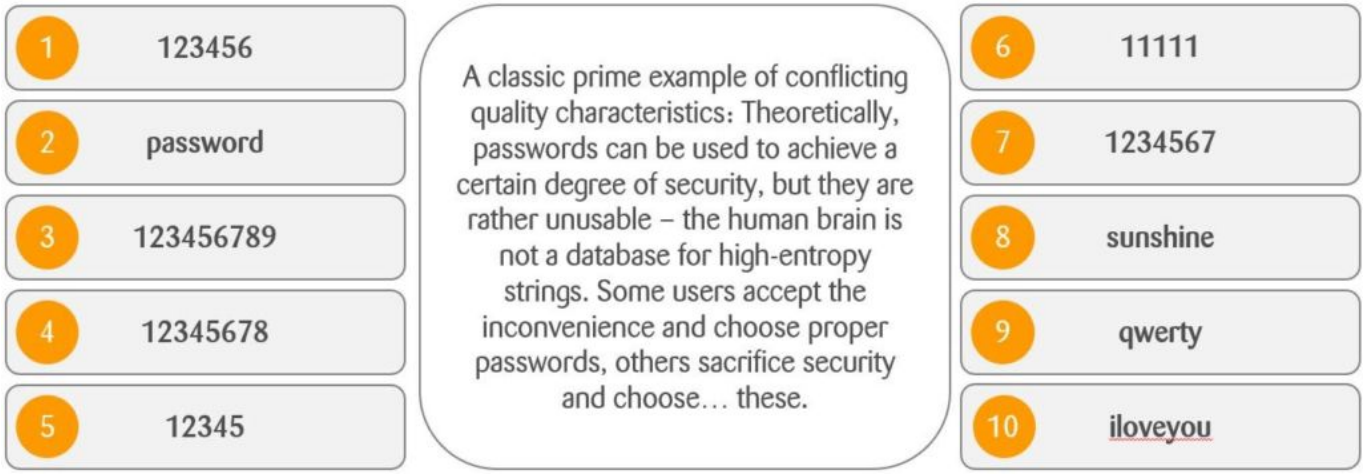

Figure 1 – Giving up security for the sake of usability

Overengineering and Gold Plating

These two terms are often used synonymously, usually when a project manager means to imply that the engineers spend too much time (and possibly other resources) on perfecting the system under development – beyond the point where it helps. A typical example is the second-system effect (a phrase coined by Fred Brooks in 1975), which is the tendency of designs of second-generation products to be exceedingly complex compared to the first generation, and hence inappropriate with respect to the required functionality.

However, we want to make a case for distinguishing between the two. More precisely, we believe that we need to consider two different dimensions in order to understand what quality is appropriate for a system under development.

The first dimension is the sophistication of the design. When a design is too trivial, this implies that it fails to provide a proper solution. Typical examples are a software architecture that fails to address some essential non-functional requirement such as maintainability or a system architecture that cannot possibly achieve the desired degree of safety (e.g. due to single points of failure). You could call such a design insufficient or underengineered. Further examples of underengineered designs are featureless architectures (suspiciously void of any product specifics), incomplete architectures (the most common case being designs that are limited to static aspects, without any consideration of the dynamics of the system), and architectures that hide the inherent complexity of the task at hand in some black box (in the sense of “and here a miracle occurs”).

The opposite extreme, of course, is overengineering: designs that are disproportionately complex and sophisticated (in software engineering, people sometimes use the term

“cathedral” to describe, or rather mock, an architecture of this kind). The most typical example for overengineering is an architecture that tries to cover all sorts of eventualities, way beyond the scope of the project. Specific examples are architectures that try to be generic platforms even though they are only expected to be used for a single, specific product and architectures that provide flexible extension points nobody ever asked for.

A special case are designs that try to answer for certain requirements by incorporating overly complex technologies. In a way, these are both underengineered (instead of providing an appropriate solution for the requirement, they simply squash the requirement by adding some heavy-weight framework or technology stack) and overengineered (because that framework or technology stack increases the overall complexity dramatically) at the same time. This may sound like an oxymoron, but it really isn’t; in fact, designs can easily be overengineered with respect to some quality characteristic and underengineered regarding another. For the sake of brevity, we won’t harp on this fact for the remainder of this post.

The other dimension is the tangible quality of the product as perceived by its users. If this perceived quality is insufficient, then the end user won’t accept the solution. Typical examples are devices that are cumbersome to operate, interactive systems that do not feel responsive, or physical products that look and feel inferior.

The opposite extreme in this dimension is gold plating: increasing the tangible quality to a point that is not expected (usually with the undesired side effect of increasing the costs beyond acceptable values). Typical examples are overly polished user interfaces for simple demonstrators or the selection of high-end parts for simple appliances. Another potential example we recently encountered is a dish washer that indicates operation by displaying a 3D animation of dishes being washed.

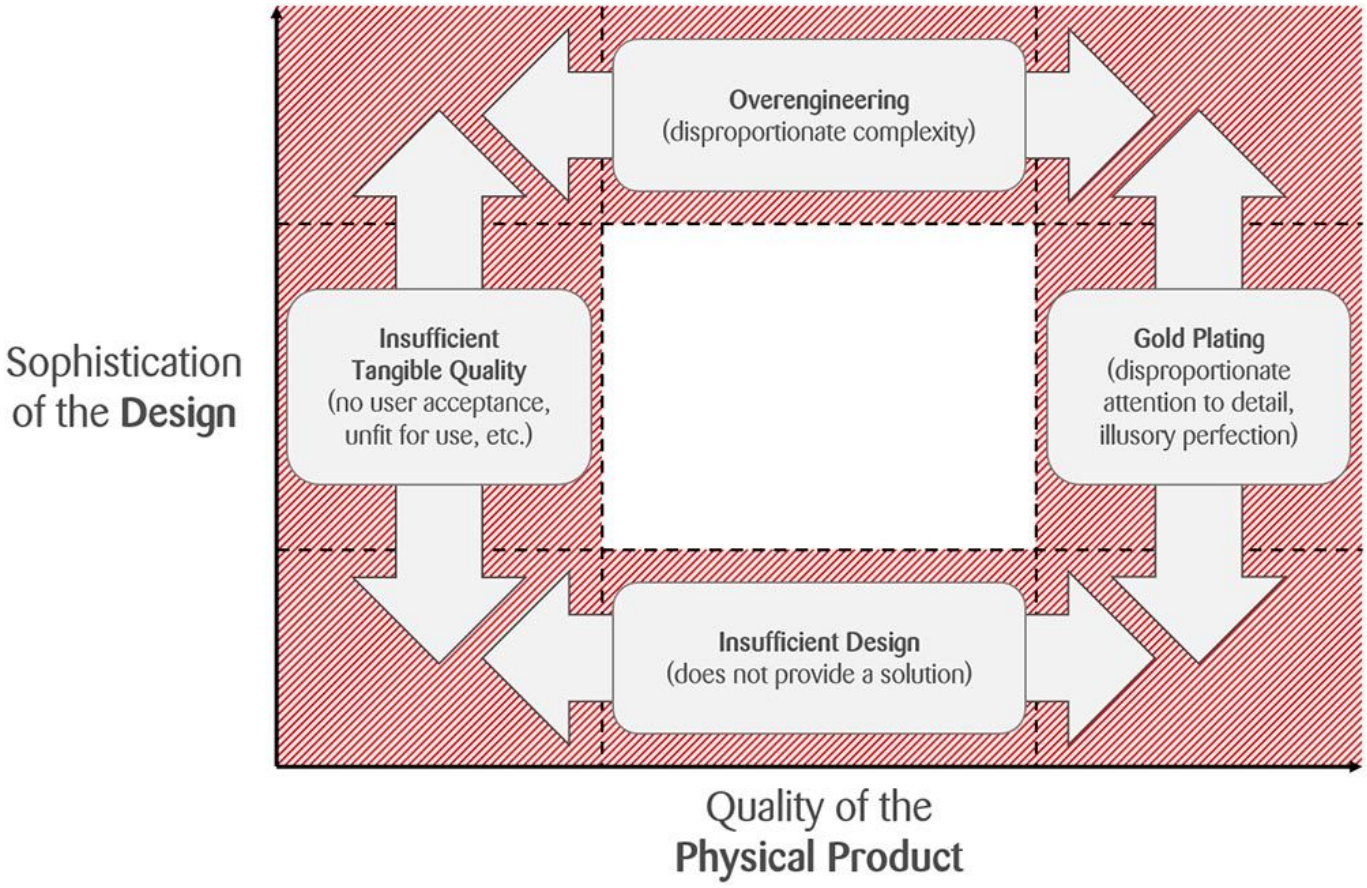

Figure 2 – The two dimensions of quality and the four extremes

It is important to note that the two aforementioned dimensions, namely the sophistication of the design and the tangible quality of the physical product, can be independent. For instance, you can easily combine a disproportionately complex design with an inferior look and feel – in the same way that some products manage to feel high-grade while being based on a design that is only waiting to break in public (just think of fancy IoT products with significant security flaws). Similarly, even products that have a flawless system architecture can feel or look cheap due to a single poor decision regarding one banal aspect such as their packaging.

So where is the sweet spot?

- We are now ready to face the crucial question: Which levels of design sophistication and tangible quality should we aim for? Let us skip the obvious (but unhelpful) answer of “It depends!” and be more specific. In fact, we can take a good step forward by distinguishing a few standard cases:
  Sometimes, especially during early development, the team needs to build engineering samples solely for the purpose of verifying that certain technical decisions suffice to solve the problem at hand. This includes so-called “spikes” or “lab samples” – constructions that don’t necessarily resemble the final product, but that eliminate technical risk by proving that a

chosen solution for a particular task of the system is sound. Since these constructions are limited to certain aspects of the system under development, they do not require a complete design. Since they are only intended for internal use, they do not require a high level of tangible quality.

- Some engineering samples are not built for the sake of development and eliminating technical risk, but for internal use by other technical stakeholders – e.g. for early testing by a verification and validation (V&V) department. These samples typically require a higher tangible quality since they are not only used by the engineers behind them.
- Another variety of output by the development team are demonstrators for use by marketing personnel or as fair exhibits. Again, these do not require the design sophistication or the tangible quality of the final product, but since they are used for demonstration and with a wider audience not comprised of tech-savvy people, they need to get close. In particular, you don’t want the system to crash in the middle of a presentation, or a user to cut their fingers on some sharp edge. On the other hand, the demonstrator only has to last for a very limited period, and it doesn’t have to be feature-complete – in opposition to the final product.
- The highest degree of maturity, with respect to both design quality and tangible quality, is obviously required for the final product.

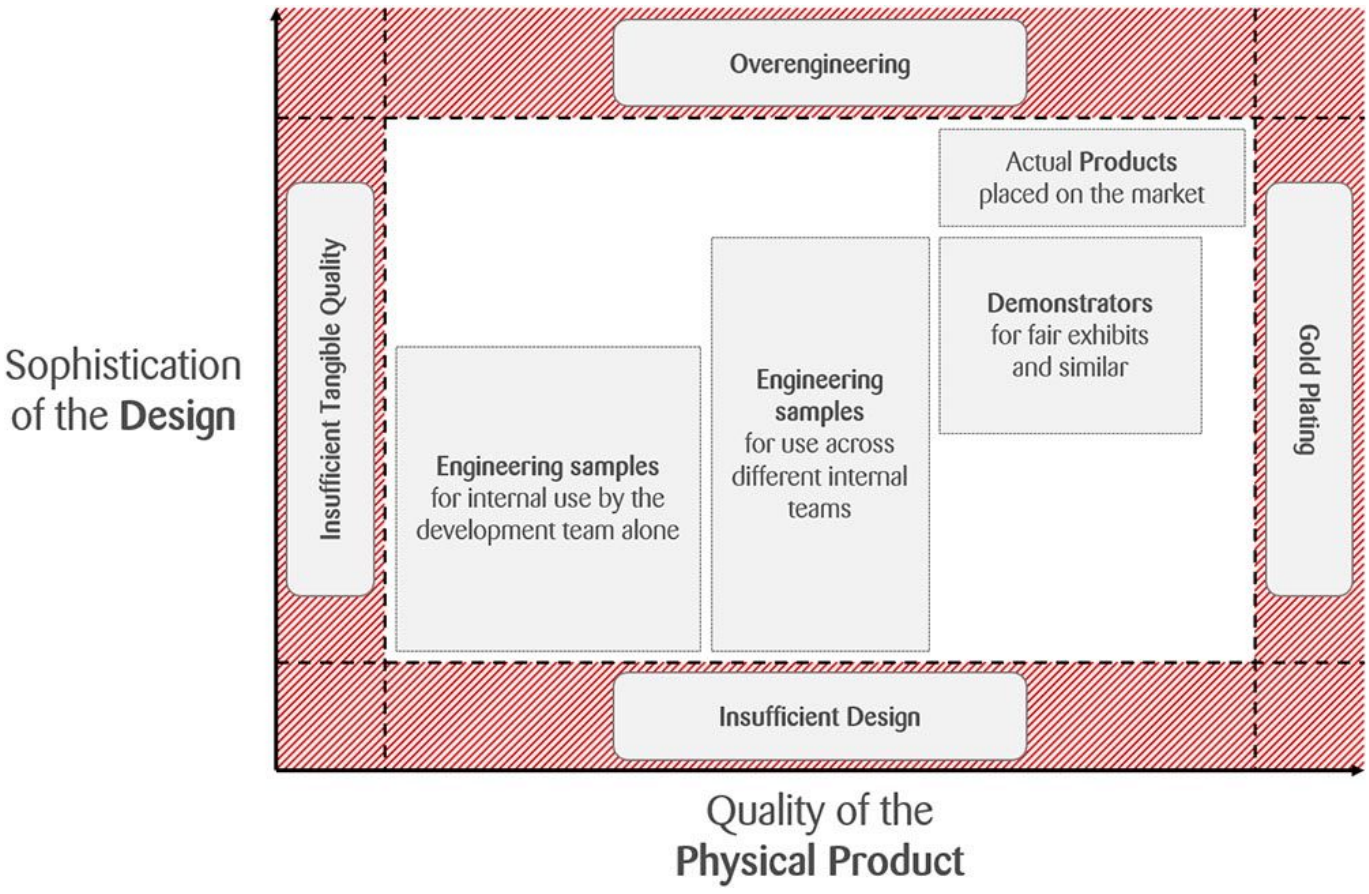

Figure 3 – A schematic and simplified distinction of target areas

Closing remarks

A typical mistake in product development is a lack of expectation management concerning the quality of the development output. For instance, consider a demonstrator: As outlined above, it may resemble the final product, but it just isn't the same thing (e.g. because the engineering step of proving that the design fulfils the desired quality levels is still pending for some characteristics). Consequently, some stakeholders may conclude that advancing from a demonstrator to the final product cannot involve any noteworthy effort. Often, this misunderstanding can only be resolved by repeated clarification.

Similarly, it is imperative to establish and maintain an explicit agreement regarding the quality levels to be achieved with the final product. You need to make sure that sponsors, product owners, and engineers alike have the same idea of what needs to be done.

Finally, we encourage you to distinguish between overengineering and gold plating. Overengineering typically happens in early phases and involves unnecessarily costly architecture/design decisions whose negative impact unfolds over long periods of time. The good news is that you can mitigate this large commercial risk by conducting an architecture assessment early on – with an excellent cost/benefit ratio. Gold plating can happen at any point of time in the project and apply to any aspect of the system. This risk is usually best mitigated by activating the maturity and discipline of the entire development team through explicit shared responsibility.