Is the Product Good Enough?
A Generic Framework for Utilitarian Quality Analysis

**GEQ Perspectives**

1. **Stakeholders:** Whose opinion about quality matters? (e.g. project team, customers, trade press, courts of law)
2. **Mission:** What do we have to achieve? (e.g. immediate survival, market share, customer satisfaction)
3. **Time Frame:** How might quality vary with time? (e.g. now, near-term, long-term, after critical events)
4. **Alternatives:** How does this product compare to alternatives, such as competing products, services, or solutions?
5. **Consequences of Failure:** What if quality is a bit worse than good enough? Do we have a contingency plan?
6. **Ethics:** Would our standard of quality seem unfairly or negligently low to a reasonable and informed observer?
7. **Quality of Assessment:** How confident are we in our assessment? Do we know enough? Is it good enough?

**GEQ Factors**

1. **Assess the benefits of the product:**
   1.1 **Identification:** What are the benefits or potential benefits for stakeholders of the product?
   1.2 **Likelihood:** Assuming the product works as designed, how likely are stakeholders to realize each benefit?
   1.3 **Impact:** How desirable is each benefit to stakeholders?
   1.4 **Individual Criticality:** Which benefits, all by themselves, are indispensable?
   1.5 **Overall Benefit:** Taken as a whole, and assuming no problems, are there sufficient benefits for stakeholders?

2. **Assess the problems of the product:**
   2.1 **Identification:** What are the problems or potential problems for stakeholders of the product?
   2.2 **Likelihood:** How likely are stakeholders to experience each problem?
   2.3 **Impact:** How damaging is each problem to stakeholders? Are there workarounds?
   2.4 **Individual Criticality:** Which problems, all by themselves, are unacceptable?
   2.5 **Overall Problems:** How do all the problems add up? Are there too many non-critical problems?

3. **Assess product quality:**
   3.1 **Overall Quality:** With respect to the GEQ Perspectives, do the benefits outweigh the problems?
   3.2 **Margin of Safety/Excellence:** By how much do we need or want benefits to outweigh problems?

4. **Assess our capability to improve the product:**
   4.1 **Strategies:** Do we know how the product could be noticeably improved?
   4.2 **People & Tools:** Do we have the right people and tools to implement those strategies?
   4.3 **Costs:** How much cost or trouble will improvement entail? Is that the best use of resources?
   4.4 **Schedule:** Can we ship now and improve later? Can we achieve improvement in an acceptable time frame?
   4.5 **Benefits:** How specifically will it improve? Are there any side benefits to improving it (e.g. better morale)?
   4.6 **Problems:** How might improvement backfire (e.g. introduce bugs, hurt morale, starve other projects)?
About this Framework

This analysis framework represents one of many ways to reason about Good Enough quality. It’s based on this assertion:

A product is good enough when *all* of these conditions apply:

1. It has sufficient benefits.
2. It has no critical problems.
3. The benefits sufficiently outweigh the problems.
4. In the present situation, and all things considered, further improvement would be more harmful than helpful.

Each point, here, is critical. If any one of them is not satisfied, then the product, although perhaps good, cannot be good enough. The first two seem fairly obvious, but notice that they are not exact opposites of each other. The complete absence of problems cannot guarantee infinite benefits, nor can infinite benefits guarantee the absence of problems. Benefits and problems do offset each other, but it’s important to consider the product from both perspectives. Point #3 reminds us that benefits must not merely outweigh problems, they must do so to a sufficient degree. It also reminds us that even in the absence of any individual critical problem, there may be patterns of non-critical problems that essentially negate the benefits of the product. Finally, point #4 introduces the important matter of logistics and side effects. If high quality is too expensive to achieve, or achieving it would cause other unacceptable problems, then we either have to accept lower quality as being good enough or we have to accept that a good enough product is impossible.

The analysis framework (p. 1) is a more detailed expression of the basic Good Enough model. It is meant to jog your mind about every important aspect of the problem. To apply it, think upon each of the GEQ Factors in light of each of the GEQ Perspectives. This process can be helpful in several ways:

1. **Use it to make a solid argument in favor of further improvement.** For instance, you might apply the stakeholder and critical purpose perspectives to support an argument that a particular packaged software product under development, while possessing cool features that will please enthusiasts, does not possess certain benefits that mainstream customers require (e.g. convenient data interchange with Microsoft Office). Mainstream customers may also require higher reliability.

2. **Use it to explore how to invest now to support higher standards later.** If you know at the beginning of a project that there will be tough quality decisions to make at the end, you can work to assure that the quality bar will be set high. Looking at the framework, you can see that by lowering the cost of improvement, it may be less of a burden and can go on longer. Preventing problems could cause higher quality to be attainable in the same time frame.

3. **Use it to form your own notion of acceptable quality.** There’s nothing sacred about this framework. It’s a work in progress. Hold your idea of quality as clearly as you can in your mind’s eye, then run through the framework and see if you find any of the questions jarring or unnecessary. Try to trace the source of your discomfort. Do you prefer different terminology? A model that more closely fits your technology or market? Are there any missing questions?

Why “Good Enough?”

Software quality assessment is a hard problem. Although there are many interesting measurable quality factors, there is no conceivable single measure that represents all that we mean by the word quality. Since quality is multidimensional and ultimately a subjective idea, a responsible and accurate perception of it must be constructed in our minds from all the facts and perceptions. It’s a cognitive process akin to analyzing the stock market, or handicapping racehorses.

When it comes to maximizing software quality, we have another hard problem—how good is good enough? Quality is not free, we have to exert ourselves to achieve it. At what point does it make more sense to turn our attention from improving a particular product to shipping that product, or at the very least, improving something else? How best can we motivate management to invest in processes and systems that lead to higher quality for less effort? We can strive for perfection, but what if we run out of time before we achieve that worthy goal? Wouldn’t it be helpful to form an idea of good enough quality, just in case perfection proves itself to be out of reach? We also need to consider that “as good as we possibly can do” might not be good enough. Even perfection might not be good enough if we seek to achieve something that’s impossible to begin with. No matter what we want to achieve, it sure comes in handy to consider the dynamics of required quality vs. desired quality.
(Opposing) Peopleware, p.151

(opposing) Microsoft Secrets, p. 43 (zero defects, quality first)

Microsoft Secrets, p.325