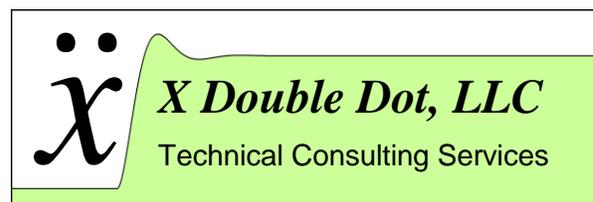


# How to KISS: The Art of Keeping it Simple, Stupid

---

By: Byron Smith  
23 April 2012

723 W Aspen Ave, Ste. #A  
Flagstaff, AZ 86001-5371  
Phone (928) 225-0788



## Introduction

*Keep it simple, Stupid.* The phrase is widely known among engineers and designers, and embodies a design philosophy known as the K.I.S.S. principle. The principle asserts that the best designs are usually the simplest and hence unnecessary complexity should be avoided. The origin of the phrase is credited to “Kelly” Johnson<sup>1</sup> while he was head of the Lockheed “Skunkworks,” a group remarkably effective at the rapid development of innovative aircraft.

Although the principle is widely known and embraced, examples of overly complex designs are not hard to find. It seems likely that in most cases the creators didn’t intentionally set out to make something unnecessarily complicated, but rather that the principle can be difficult to reduce to practice. This paper addresses the question of what can a design leader do to produce simpler and more effective designs.

“Everything should be made as simple as possible, but not simpler.” – Albert Einstein  
[attributed]

## KISS Principle

Let us begin with some consideration of our goal and what it means to be simple. Certainly, the advanced supersonic jets produced by the Skunkworks were not simple in any absolute sense. Rather, the KISS principle is to keep designs as simple as possible, by avoiding unnecessary complication. To break that down further, we can consider the following definition:

### **Complicated**

1. Consisting of parts intricately combined
2. Difficult to analyze, understand, or explain

So, we are seeking designs using the fewest components and interrelations between those components as possible. And, it should be easy to comprehend and communicate both the design itself and how it works.

These two aspects are both important in supporting our ultimate goal of the creation of designs which work reliably while expending a minimum of time and effort. The reduction of components and interfaces eliminates the associated need for design, drawing checks, manufacturing, inspection, rework, assembly, test, spares management, etc... This reduces both effort and opportunities for mistakes. Likewise, ease of comprehension and communication improves the ability to coordinate the individuals involved in collaborative development efforts. Conceptual simplicity reduces the effort involved with analysis and documentation, and associated opportunities for mistakes and misinterpretation. Ease of comprehension and communication becomes increasingly important with larger development teams.

## Cultivate Simplicity

The foundation of realizing simple designs is a culture of simplicity. That is, a design environment in which every team member knows that simple solutions are expected. The design leader must cultivate this by explicitly stating that objective. As an example, consider the following description of the Skunkworks mission<sup>3</sup>:

“Our aim, is to get results cheaper, sooner, and better through application of common sense to tough problems. If it works, don’t fix it.” – Kelly Johnson

This mission statement seems to have worked for the Skunkworks, and might well be adopted verbatim by many design groups. It may be also be worthwhile to formally consider simplicity in the design process, e.g. as a criterion for concept trade studies.

Beyond recording a mission statement, a culture of simplicity requires the philosophy to be a part of everyday operations. At the Skunkworks, this was served by the mantra of “Keep it Simple, Stupid.” The author once worked in a design group where one would be gently admonished as “Rube” (Goldberg) for presenting an overly-complex design concept. However the philosophy is expressed, a team striving for simple designs and its members reminding each other of that goal is the essence of a culture of simplicity.

The challenge of developing such a culture shouldn’t be underestimated. Engineering is a discipline filled with problem-solvers; and the problems are usually presented as matters of expanded functionality and performance improvement. Further, as tools and techniques for complexity management continue to improve, risk compensation behavior often leads to more complexity.

## Keep Requirements Simple

The complexity of a design starts with the requirements, and the easiest ones to satisfy are those that don’t exist. Accordingly, some effort on keeping requirements pared to the minimum can yield benefits. Basic requirements management practice is to be wary of creeping elegance in the form of numerous little additional requirements, and that stakeholders must consider the cost vs. benefit of proposed changes. While the theory is sound, the practice faces the fundamental problem that costs and benefits are difficult to quantify accurately during the requirements phase. Further, we tend to have an optimistic bias for the direct costs that we anticipate and often neglect indirect costs and unanticipated expenses. Unfortunately, the assessment of benefits is often optimistic as well. To help address these biases, requirement managers can set a minimum expected return on investment for requirements. This minimum should depend on the historical accuracy of predictions, but at least 2:1 would be a reasonable start.

While top-level requirements (especially those associated with contracts) usually face scrutiny, the flow-down of requirements to components is often a more lax process and an easy place to accrue complexity. In the absence of a formal systems engineering process, all flow-down requirements should at least have a rationale associated with them. This helps avoid the tendency to over-specify

components when the actual requirements aren't known or understood. It's better to spend a little effort on understanding what *is* needed, than a lot of effort on providing capability that *might be* needed. Also, emergent requirements should be addressed early. If a design utilizes an emergent sub-system or process (e.g. a computer system, an energy source, a preventative maintenance program, etc...), don't let the burden of developing the requirements be deferred. The consideration of the emergent complexity needs to be a part of the initial design trades.

## Architect Better Comprehension

For large and complex problems, good architecture improves the comprehensibility of design solutions. While this may not reduce the total number of components used in the design, conceptual simplicity is benefitted by organizing the components in ways easier to analyze, understand and explain.

As the number of components,  $N$ , in a design increases, the number of possible interactions between those component increases as  $N(N - 1)$ . The potential complexity thus grows very quickly, and is known as the N-squared problem. To address this problem, designs may be architected to aggregate groups of components together which interact heavily with each other, and partition those groups such that the overall number of interactions is minimized. In software engineering, this goal is referred to as having high cohesion of components within groups and low coupling between groups. The effect is to break a complex system down into a number of more easily understood sub-systems, with easily understood interactions between them.

As a system is broken down, each sub-system may be considered its own project. The system should be divided such that these sub-projects are matched to the groups executing them. E.g. they may still be very complex for a sub-project contracted to a multi-national company, which will further divide the project within its organization. The practical atomic division is probably a level that would be comfortably executed by a small well-coordinated design team.

## Strive for Ultra-reliability

Ultra-reliable components just work. Ultra-reliability is a term related to ultra-quality<sup>4</sup>, which is a level of quality so high that defect rates are impractical to measure. Reliability is an aspect of quality, defined as the probability that a component will perform its function over a specified life. Ultra-reliability then implies failure rates that are impractically low to measure, but more importantly are low enough to justify not worrying about failures in the design and operation of a system.

Ultra-reliability can free a design from a host of contingency requirements, such as redundancy, service provisions, maintenance procedures and training, spares management, etc... By striving for ultra-reliability, the design leader shifts focus from making contingency plans to avoiding the need for those plans.

One method of increasing reliability without increasing complexity is by derating. The failure rate of many components is related to the relative stress seen in service. Often a given derating can achieve a proportionately larger reduction in failure rate. E.g. derating a ball bearing capacity by a factor of five will reduce the failure rate by just over an order of magnitude. Certainly, derating has its fiscal and performance costs, but balance those against the overall logistics costs of addressing failures.

Another effective way to increase quality without complexity is to make tolerances as generous as possible. As quality is measured relative to requirements, the metric can be improved by increasing performance (expensive and hard) or lowering requirements (cheap and easy). In any given design there are a few critical parameters and many that are non-critical, but which are critical may not be obvious in the initial design. The tendency is to assume typically achievable values for all parameters for the initial analysis, and tighten the critical values as necessary. The analysis assumptions form the basis of subsequent requirements and the typical values aren't usually challenged. However, specifying typically achievable values (met most of the time) is inconsistent with ultra-quality (requirements met nearly all of the time). The design leader should then start a design with tolerances that are rarely exceeded. This might be assumed as twice the typical values ( $6\sigma$  vs.  $3\sigma$  for a normal random process) in the absence of better information. The critical parameters will often require extraordinary attention, but there is no point expending extraordinary effort on non-critical parameters.

Lastly, ultra-reliability can be used in conjunction with architecture for complex systems. Requiring sub-systems to be ultra-reliable may necessitate additional complexity, such as redundancy or additional quality control. However, any additional complexity is isolated within the sub-system and simplifies the system-level view. There are many trade-offs between requiring ultra-reliable sub-systems and accommodating failures at the system level, and the design leader should consider both options.

## Don't Forget Analog

Analog still works. Although analog and discrete digital approaches don't receive marketing attention like the latest programmable devices, they can provide very effective solutions to simple problems. Programmable devices can handle very complex problems with simple hardware, but introduce significant software development and operational complexity. Programmable devices certainly have their place, but design leaders should seek to match the complexity of solutions with the problems.

To minimize complexity, consider the following guidelines: If it is practical to fulfill a design goal passively, do so. If passive means won't suffice, then if practical use active analog and discrete logic. Otherwise, employ a programmable device with the simplest software (development, application and OS) possible.

Analog has other benefits for interfaces, as physical units are ubiquitous, non-proprietary and obsolescence proof. E.g. the Volt was defined in the late 1800's, and is an open-standard which isn't going anywhere soon. Digital communication may offer potential performance and functional advantages, but often imposes a number of implicit requirements on both sides of the interface.

## Conclusion

Although the KISS principle is widely known and cited, literature on how to put the principle into practice is limited. Engineering efforts and discussions seem much more focused on achieving better performance at the expense of complexity, than achieving similar performance with reduced complexity.

Rather than to be definitive, this article is intended to promote thought and discussion of how to keep designs simple. The author is interested in hearing from readers who have other ideas on how to embody the principle in practice. Please feel free to contact him at:

[bsmith@xdoubledot.com](mailto:bsmith@xdoubledot.com)

“It seems that perfection is attained not when there is nothing more to add, but when there is nothing more to remove.” – Antoine de Saint Exupery [translated]

## References

- 1) Johnson, C., Smith, M., *Kelly: More Than My Share of it All*, Washington: Smithsonian Institution, 1985.
- 2) “Complicated.” *Merriam-Webster’s Unabridged Dictionary*, 2012.
- 3) Rich, B., *Clarence Leonard (Kelly) Johnson: A Biographical Memoir*, Washington: National Academies, 1995.
- 4) Rechtin, E., *Systems Architecting, Creating and Building Complex Systems*, Englewood Cliffs, NJ: Prentice Hall, 1991.