Lean Efficient Software, less is Moore

Daniel S. Fowler (ORCID: 0000-0001-6730-2802), dan@tekeye.uk
Tek Eye UK

When Developing Software and Systems Follow Chuck, Be as Lean as Possible

Why is developing software and designing systems expensive? It is because the Information Technology (IT) industry has failed to embrace the KISS principle [1] (Keep it simple, stupid!). Computers, software and the tools to develop them have followed an upward complexity trajectory for many years. The complexity in software and systems kicked in when the Graphical User Interface (GUI) became commonplace. All developers and designers in the computing field should look at the path followed by the pioneering engineer Charles Havice Moore, known as Chuck Moore. You could consider his family name and his design principles an oxymoron; for Moore his lifelong ideal was to design less.

Lean, Mean Computing Machine

In the 1981 film Stripes the character of Ox, played by John Candy, has joined the army to lose weight and become a *"lean, mean fightin' machine"* [2]. Many systems suffer from bloatware and over-engineered software and would perform much better if they lost their complexity, lost their *weight*. Regularly there are reports of multi-million, even multi-billion dollar, computing project failures, whether hardware or software or both; complexity is often given as one of the reasons for failure. You can also look at the disappointment consumers often experience with electronic devices they buy. They are full of extraneous software and processes that eat processor cycles and computer memory causing poor interface response times and system crashes. Software and computer hardware have a bad reputation yet both are important areas of modern engineering, the modern world relies on computer systems. In some ways, the move to agile development methods was a response to the increase in system complexity. A way to try and cap the ever-increasing costs of developing modern systems. Yet Chuck Moore foresaw this early in his career and has always advocated the need to remove complexity from design; that way you still get a working system but you get it quicker and cheaper.

Beware the Ubergeek

In most computer projects there are key personnel who are largely responsible for the design, code and build of the system. These individuals tend to be very intelligent, work quickly and are highly productive. They are the company gurus or ubergeeks. The rest of the team usually acts in a supporting role, implementing their decisions, doing the grunt work, fixing errors, running tests, shipping builds and authoring the documentation. Whilst organisations tend to rely on the ubergeek for most of the productivity there can be a downside that can cause long-term maintenance problems. Problems that can affect the ongoing profitability of a project. If not carefully managed ubergeeks can over-design with unnecessary complexity. In code, this is often manifested as complex object-orientated models, in hardware too much redundancy and too many components. They often cite reuse-ability as the argument for their designs, however, very few designs get reused again. Only the ubergeeks understand the design and they are doing it because of their deep understanding of the tools and techniques being used. The rest of the team then struggle to understand the complex code and designs. The work they do on the system then gets criticized by the ubergeeks for not fitting in with the model they have built. This can lead to team discord. Further down the line, with the system delivered to the customer, the ubergeeks move on to their next project. They have no interest in their previous projects, they want to play with new tools and techniques. The project is in maintenance but no one understands the intricate design. This leads to patches to the system that adds further complexity, the code in the system becomes tangled and unstructured, becoming spaghetti code. The extra complexity often increases the cost for what should be small and inexpensive changes, reducing the forecasts for the profitability of the project, and if the margins are small, pushing it into loss-making.

The spaghetti image is CC by SA [3] from Wikimedia Commons [4].

Keep It Simple Stupid

The only way to maintain project profitability is to rein in the design. That sometimes means questioning the actions of the ubergeeks and getting them to produce simpler models and straightforward code, they may be frustrated and even bored by this, if so their talents may be better employed elsewhere, e.g., research and development, and the project may require a more pragmatic programmer, just a geek instead of an ubergeek.

The *less is more* mantra applies equally to software development as much as it does to other areas of modern life. The phrase is taken from the 1855 poem *Andrea del Sarto* by Robert Browning [5] (known for his poem *The Pied Piper of Hamlyn* [6]). In the Andrea Del Sarto poem, Browning has the Italian painter lamenting his lack of ability to add depth or soul to his work, but the painter praises his own technical accuracy. However, it is technical accuracy (a crisp and clean design) that is needed in projects and no flourishes or extraneous additions, hence *less is more*. This principle was taken up in minimalist and modernist design in the 20th Century and has existed in far eastern culture way before then due to the Taoist religious beliefs. This is exemplified in the rise of lean manufacturing in Japan where efficiency is maximized and waste minimized. The same lean concepts must be applied to systems engineering. When designing software and hardware only do what is necessary and no more, and do it with straightforward code. No planning for reuse, just delivering to requirements and no more, because it is nearly always the case that those plans for reuse never match the new requirements that emerge. Whereas a simple and elegant design can be changed and adapted easily and quickly, often by those that are less skilled than the ubergeeks, and that is good for project profitability.

Simplicity in Software

Simplicity within the software, system, and programming is not a new concept. The UNIX operating system came out in the 1970s under the principle of modularity and that programs should do one thing, but do that one thing well. The Unix philosophy [7] on systems and software is encapsulated in four rules from 1978:

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".
2. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.
3. Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.
4. Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

Rule 3 is the essence of agile programming, showing that release often was a good idea before it became the normal way for successful software projects. Why embracing simplicity works can be summed up in this paragraph from the book The Unix Programming Environment [8]:

"Even though the UNIX system introduces a number of innovative programs and techniques, no single program or idea makes it work well. Instead, what makes it effective is the approach to programming, a philosophy of using the computer. Although that philosophy can't be written down in a single sentence, at its heart is the idea that the power of a system comes more from the relationships among programs than from the programs themselves. Many UNIX programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools."

Combinations of simple code can make an elegant and effective system. Look at nature as an example, each ant and bee usually has a single task, but together the colony or hive achieve great results. When it comes to software projects follow Chuck's mantra on simplicity, and remember that less is Moore.

Let's Finish with Steve Jobs on Simplicity

Steve Jobs - *"Simple can be harder than complex: You have to work hard to get your thinking clean to make it simple. But it's worth it in the end because once you get there, you can move mountains."* - From an interview in BusinessWeek, 1998 [9].

Image: Gabriel Fernandes, São Paulo, Brasil, CC BY-SA [3], via Wikimedia Commons [10]

Acknowledgements

[1] KISS Principle, https://en.wikipedia.org/wiki/KISS_principle

[2] Stripes film quotes, https://en.wikiquote.org/wiki/Stripes_(film)

[3] CC by SA, https://creativecommons.org/licenses/by-sa/3.0/

[4] Spaghetti, https://commons.wikimedia.org/wiki/File%3ASpaghetti_aglio_olio_e_peperoncino_by_matsuyuki.jpg

[5] Andrea del Sarto, Robert Browning poem, https://www.poetryfoundation.org/poems/43745/andrea-del-sarto

[6] The Pied Piper of Hamelin, Robert Browning poem, https://www.poetryfoundation.org/poems/45818/the-pied-piper-of-hamelin

[7] Unix philosophy, https://en.wikipedia.org/wiki/Unix_philosophy

[8] The UNIX programming environment, https://doi.org/10.1002/spe.4380090102

[9] Steve Jobs quote,
https://web.archive.org/web/19991110003323/http://www.businessweek.com/1998/21/b3579165.htm

[10] Museum image, Gabriel Fernandes,
https://commons.wikimedia.org/wiki/File:Museu_da_imagem_e_do_som_de_s%C3%A3o_paulo_exposi%C3%A7%C3%A3o_%E2%80%9Csteve_jobs,_o_vision%C3%A1rio%E2%80%9D_(34987316000).jpg