# “Software bloat” isn’t an endemic problem

Wentao Zheng

December 28, 2014

According to Wikipedia:

> **Software bloat** is a process whereby successive versions of a computer program become perceptibly slower, use more memory / diskspace or processing power, or have higher hardware requirements than the previous version whilst making only dubious user-perceptible improvements.

## Illusion of “user”

The first part of the statement is probably true: Newer versions of software are usually bigger in size, run slower under the same hardware / platform, use more memory / disk, and consume more CPU cycles and energy. However, the second part with *only dubious user-perceptible improvements* is questionable, because the notion “user” is not clearly defined.

Let’s take a look at the classic comparison between CLI (command line interface) and GUI (graphical user interface). In the early days of computer, operating systems didn’t have fancy UI, and were only used by scientists and engineers, for whom CLI was powerful, flexible and elegant enough to perform most daily tasks. With the popularity of PC (personal computer), GUI replaced CLI as the “standard” way of human-computer interaction. In terms of functionality, it’s hard to find anything that GUI can do but CLI can’t. If we define the “user” as an engineer, it is probably true that GUI only provides “dubious” improvements on top of CLI. However, if we define the “user” as a non-technical person, CLI becomes much harder to use than GUI. Learning common commands takes time and mastering them is very difficult for people without any technical background. In comparison, GUI is more intuitive and easier to master. Here is an example: My two-year old daughter can watch YouTube videos and navigate camera photos with my iPad. How can you imagine such thing could happen if we were living in a CLI world?

I’m using the transition from CLI to GUI just to demonstrate software *user* is not a static thing. Its definition changes as software evolves. Its scope is usually expanding, because of business reasons (growth, growth, and more growth!).

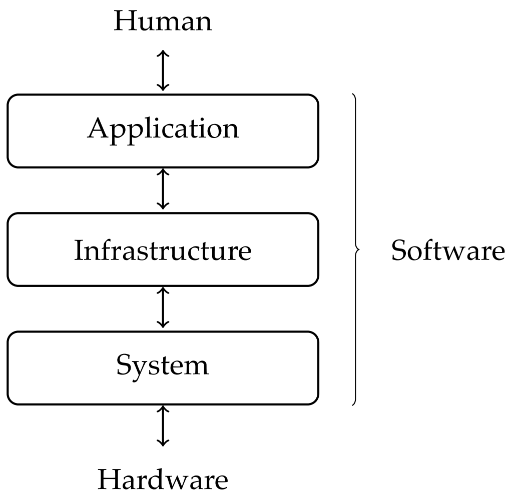
The first user of software is usually the author / programmer himself (let’s assume a male engineer here). He builds it, tests it, and fixes all the bugs he ever finds. When he thinks his software is in a good state, he probably will send his “perfect” work to the second user: his boss in most cases. Feedbacks are inevitable, changes and improvements shall be done. Then the software will be released to the market. More feedbacks are coming in, more changes and improvements have to be done. This cycle repeats, and we will have a side effect: the software ends up having so many features that nobody would ever use 100% of them. Earlier users won't use features needed by later users; later users need new features because some features provided in earlier version of software are useless to them.

Now let's look back at the definition of "software bloat": Is it really a problem? Probably not at all. Even if you think it's a bad thing from software engineering perspective, I don't think this kind of problem can be solved.

## Theory of evolution

Software adapts to the changes of the user. The user is not always human. Sometimes, its user is software itself. Examples of software being used by other software include libraries, frameworks, and platforms. Software also adapts to the changes of what it is using, which are called dependencies. The universe of software is similar to an organic ecosystem. Everything has to adapt to changes of its surrounding environment: either its users or what it is using.

The dependency chain in the software universe is like the food chain in an organic ecosystem. It is a layered structure. At the bottom, we have *system software*, such as compilers and operating systems, which are responsible for hardware-software communication. On top of that, we have *infrastructure software*, such as database, which is used by other software. At the top, we have *application software* that is used by human users.



The ecosystem's external changes come from two sources

- Hardware: faster CPU, faster disk, more memory etc.
- Human: requirement changes, user scope expanding etc.

These changes will trigger reactive responses from the internal software universe in both directions: bottom up and top down.

As a software developer, you have two choices when facing those changes: Carefully plan improvements to adapt them and make your software better in terms of survivability; Or stick with the old “users” and avoid being a bloat-ware developer. It’s very hard to tell which choice is better. It depends on the situation. Under the first choice, no matter how careful you are, adding more features in software brings in complexity and bugs. That could kill your business if it’s not executed properly. Under the second choice, staying unchanged means losing opportunities. New software might come out replacing yours in the near future.

Remember backward-compatibility is like a curse to software. When you see changes coming, you are forced to add more stuff into your software because you don’t want to lose older users or functionality. In this sense, software has to “bloat” in order to survive into the future.

## End

I would like to conclude: “Software bloat” is not a problem, but a inevitable destiny we as developers have to face. We want to make sure our software grows in order to adapt changes. Failing to do so with proper engineering practice could kill your software (or your business) in the future.