“Software bloat” isn’t an endemic problem

Wentao Zheng

December 27, 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 the 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 something 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, when the “user” becomes a non-technical person, CLI is almost useless. 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. The definition of the user changes as software evolves. The scope of the user 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 found. When he think it is in good state, his “perfect” work will be reviewed by his boss, the second user. Feedbacks are inevitable, changes and improvements shall be done. Then it goes into the market. More feedbacks are coming in, more changes and improvements have to be done. This cycle repeats. A side effect is: software ends up having so many features that nobody would use 100% of them. Earlier users won’t use features needed by later users; later users need new features because 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 changes of the user. Sometimes, software uses software, which are called libraries, frameworks, or platforms. They adapts to changes of end-user facing software. The universe of software is similar to an organic ecosystem. Everything has to adapt to changes of the surrounding environment: both its users and what it is using.

The dependency chain in software universe is like food chain in organic ecosystem. It has a layer-like structure. At the bottom, we have compilers and operating systems, which are responsible for hardware and software communication. On top of that, we have many infrastructural components that are not used by end-user directly, but by other software, such as database. At the top, we have application software that is used by human.

The external changes come from two places:

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

These changes will trigger reactive response from the internal software universe in both directions.

As a software developer, you have two choices when facing those changes: Carefully plan to adapt those changes, make your software better in terms of survivability; Or stick with the old "users", and avoid to be a bloat-ware. It's very hard to tell which choice is better. It depends. 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 usually add more stuff into your software. In this sense, software has to "bloat" in order to survive into the future.

End

"Software bloat" is not a problem, but a inevitable destiny we have to face. As a developer, you want to make sure your 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.