Software: Gradually and Then Suddenly

Daniel "Drex" Drexler
Center for Science, Technology and Society at Drexel University

Software: Gradually and Then Suddenly

Gradually and Then Suddenly

The Greek idea of the atom was that it was the final, indivisible unit of matter. Their mistake is understandable: atoms are only split with great difficulty. Atoms and software are both encountered as singular entities, but their singular qualities emerge from their parts. With atoms those parts are quarks and leptons. Software is built from libraries (themselves also software) and source code. To properly appreciate how software comes to behave, we have to both talk about the "atomic" software (indivisible, assembled, static) and its components.

This text, appearing as it does on the screen or page you are reading, is the result of a chain of software assembly processes. Almost every piece of software relies on a long list of chains of assembly processes. A piece of software is assembled[1] together from its component parts: source code, already assembled software, data, and the many tools that will actuate the assembly[2]. Each piece of software involved in this assembly process was itself assembled in its own process of gathering together. Each software has its own way of assembling its components. Different softwares use different kinds of tools (and each kind also has a diversity of particular options). The set of tools used by a piece of software and the way those tools are configured is part of the existence of that piece of software. A piece of software has source code that is (before it is assembled) solely used by it, but the other pieces of software (both those responsible for providing functionality and making the assembly) are equal partners in the constituting process. The software cannot be assembled if any part fails. The qualities of the software can be traced back to any of these components and in complex software behavior is often the result of all of them working in concert.

The product of an assembly also varies from software to software. Sometimes the assembly produces a single binary file that can be read and executed by one or more kinds of computer. It can also produce a particular arrangement of components that will be interpreted by an already existing binary. The commonality between these approaches is they produce a constellation of resources that, for the user of the software, function as a single "atomic" piece of software. For most people who interact with software, they experience it as this atomic whole: a cohesive thing that acts with one mind. When seeking to understand *why* a particular piece of software does what it does, however, it is important to pull apart this atomic structure. Much of what an app on your phone does is dictated not by the source code files of the app, but by the operating system of the phone itself and the associated world of shared libraries that the app utilizes (either because the phone's operating system requires it or because the library saves the app maker time).

The nature of software is important to understand before discussing how

---

[1] This is distinct from a computer assembler (see next note).

[2] Astute readers will have realized that this begs the question a bit. The tools that actuate the assembly are software. Making software involves software. When we go all the way back before the assembly process, we find programs written directly in the language understood by computer processors - machine language. Software must always make its way back to the machine language for the machine it is running on somehow. It's the only thing computers speak. The tools that accomplish this have various important qualities that are unimportant to the work here. They include assemblers, compilers, and interpreters. The important quality remains that they are all *texts*. Instructions for machines written by humans.

"software" "works" because that discussion may, at any point, need to break off to explain that the relevant piece of software has suddenly changed. The text messaging app on your phone relies on your phones operating system to send a receive text messages, which in turn relies on the software of the phone's modem to translate texts to radio waves. When a problem arises, it can arise within any of these components, in the interactions between these components, or when the process executes correctly but the user expected a different result. Though it is possible to draw boundaries around all of the source code involved in a single piece of software, it quickly becomes enormously difficult. Facebook's website uses just over sixty million lines of source code, but it also relies on behavior that exists in the operating systems it uses (linux [over ten million lines] or windows [over forty million]) and the source code that runs each component of the computers it uses (McCandless, 2014). When a user encounters a problem in "Facebook" the problem is most likely in the source code written by Facebook, but it may not be! Understanding this chain of agency within computers is essential to tracking why software does "what it does."

Eden

Emergency Development ENvironment (EDEN) is "an open source software platform which has been built specifically to help in Disaster Management[sic]" (Sahana Foundation, 2011). As an atomic piece of software users experience EDEN as a series of web pages that focus on managing resources. The resources available to manage are diverse: people, goods, events, goals, processes and locations. The Sahana Foundation, the organization most responsible for EDEN presents these capabilities as focusing on managing the sprawling nature of the world and keeping multiple things within the bounds of perception of the users of the system (Sahana Foundation, n.d.). Their pitch focuses on the needs and concerns of the officials selecting the system, but it also takes into account the need for individual members to manage and access the knowledge that the system manages. The the Sahana Foundation sketches an image of EDEN organizing and marshaling limited resources in chaotic times.

Glossary

source code Human recognizable language, generally written in primaraly latin text in sets of files. Examples include C, Python (the language of EDEN), LaTeX (the language this paper is assembled by).. 2, 3

Acronyms

EDEN Emergency Development ENvironment. 3

References

McCandless, D. (2014). *Knowledge is beautiful*. William Collins.

Sahana Foundation. (n.d.). */chapter: What-Does-Sahana-Eden-Do / sahana eden.* `http://write.flossmanuals.net/sahana-eden/what-does-sahana-eden-do/`. (Accessed: 2020-2-20)

Sahana Foundation. (2011, December). *Sahana eden brochure.* Online.