What is a Software Engineer?

Al Kaniss, Naval Air Systems Command Headquarters

Many years ago, at a Software Technology Conference, one of the briefers talked about how creating computer code had gradually evolved from art to craft to engineering. The briefer had a slide with a graphic of a doghouse, a typical suburban single-family house, and a skyscraper. His point was that the difference between the three structures wasn't just the progressively larger size, but rather the vastly increasing complexity. The briefer explained that your teenager could probably successfully build a doghouse, but couldn't build the family house. And the builder who built your family's house most likely couldn't build a skyscraper. Software Engineering is exponentially more complex than cobbling code together which satisfies a functional requirement.

That mental model of size versus complexity stuck with me ever since. In fact, I use a similar graphic (figure 1) three times a year, when I brief new NAVAIR Systems Engineers about software and software engineering. People still tend to equate software engineering with computer programming. While programming, i.e., writing computer code, is certainly an important facet of the software engineer's duties, it may be as little as 20 to 30 percent of those duties.

Yes, in the early days of software development, things were vastly different. Computers didn't have much memory, so programs couldn't be very large. Computers weren't networked nor even accessible by the average person. They were either in large, locked rooms tended to by operators in white lab coats, or in specialized labs associated with weapons systems. Just entering a small program by setting switches, observing lights and getting it to run successfully was a major accomplishment.

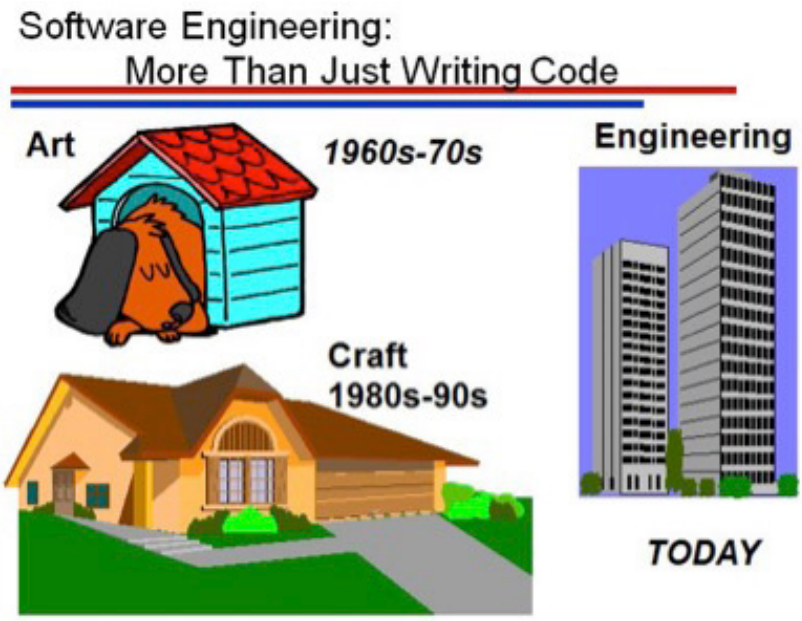

Figure 1

Contrast that with today's environment. The majority of people on the planet have access to a fairly powerful computer with access to most of the other computers on the planet via the Internet. Cybersecurity, long ago not much more than preventing access to a locked room and changing one's password periodically, has become one of the major responsibilities of a software engineer.

Along with increased complexity, software development also got more disciplined. Back in the early days, a software developer just kept typing code and running it until the desired result was achieved. For small, simple computer programs, this was time consuming but adequate. As computer capacity grew and grew, and teams of developers replaced the lone computer programmer, more discipline was required. Requirements, designs, integration plans and tests had to be formally generated, documented, tracked and traced to assure that the whole team was in sync with a high probability of an integrated, successful product, especially as we've gone from computer programs of hundreds of lines of code, to suites of software for systems of over 24 million lines of code. SEI's CMM® and the CMMI® were created out of necessity for increasing the discipline of software development teams and became commonly used by organizations large and small.

The goal of those involved with software development has never changed: creating a reliable, functional computer program. The responsibilities of people who are involved with doing so, however, have grown enormously. Software Engineers are fully engaged with Systems Engineers to decompose system requirements into software requirements. Software architecture and design have become increasingly important, especially as systems have been networked and have become "systems of systems". The tasks of software integration and system integration have grown exponentially as the number of computer modules (CSCIs, CSCs and CSUs) and subsystems grows with computer capacity.

Software safety has also become an increasingly significant responsibility of Software Engineers over the past 25 years, as software has been given increasing control over systems. As mentioned, Cybersecurity is also a major concern these days. Regardless of how well software functions, if it is not protected from its threats, its value diminishes greatly.

Software Engineers must also create and maintain software that satisfies a lot of other demands. Since the life of software can be 10, 20, 30 or more years, it must be designed to be easily re-hosted on newer hardware without major (and thus costly) changes. Software must also be easily modifiable over its life. We've all heard of "spaghetti code", which can be more difficult and costly to modify than it was to create, especially if the people maintaining the code had not written it in the first place or lack adequate documentation detailing it.

Another attribute that people demand of software these days is that it be reliable. This can cause a lot of confusion as software is always reliable, in that it doesn't break, wear out or rust out like hardware does. The software however operates within a system, which is really the entity that must be reliable. The state of the computer (including other software executing and operator and other external inputs) can make the software appear unreliable. It is up to the Software Engineer to design the software to be tolerant of such things.

Users of computer software want it to be "user friendly". User friendliness is of course an ambiguous requirement, and if you ask

100 users how they want some piece of software to look and behave, you will likely get 100 different answers. And likewise, as users get more familiar with the software, they want the user interface to grow from "beginner mode" to "expert mode", with fewer prompts and more complex screens as their expertise grows.

Complicating everything else, there is more and more pressure to field software more quickly and thus more cheaply, requiring Software Engineers to increasingly learn and use Agile methods. We want software that is of high quality, produced quickly, and at minimal cost. That is quite a tall order. Add to those other attributes we require of software (figure 2), the Software Engineer has the Herculean task of satisfying all the people all the time. And often, such attributes conflict. For example, making a system "open" to decrease costs and facilitate software re-use conflicts with making a system secure.

The explosive growth of software-reliant systems vastly increases the need for talented software engineers. Hopefully, as time goes on, we will continue to develop enough people who have the full complement of skills necessary to accomplish such work and attract them to work in the Defense environment.

Ironically, there is no Software Engineer title in the federal government. We hire people into the existing Computer Engineer, Electronics Engineer and Computer Scientist billets. Hopefully someday soon, such a title will exist. It's also ironic that in the early days of Software Engineering, one of the newer engineering fields, some of the more traditional types of engineers (civil, mechanical, electrical) tended to challenge the notion of a "software engineer", since it didn't involve physical things like buildings and bridges. Hopefully, that opinion is long past as software has become so critical a component of virtually every system that is produced today.

Disclaimer:

CMMI® and CMM® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

ABOUT THE AUTHOR

Al Kaniss has worked for the Navy for 39 years in various capacities as a Software Engineer...long before "Software Engineer" was even a common term. He is now Branch Head for Software Engineering at Patuxent River Maryland, home of Naval Air Systems Command Headquarters.

Category	Quality attribute	Description
Design Qualities	*Conceptual Integrity*	Conceptual integrity defines the consistency and coherence of the overall design. This includes the way that components or modules are designed, as well as factors such as coding style and variable naming.
	Maintainability	Maintainability is the ability of the system to undergo changes with a degree of ease. These changes could impact components, services, features, and interfaces when adding or changing the functionality, fixing errors, and meeting new business requirements.
	Reusability	Reusability defines the capability for components and subsystems to be suitable for use in other applications and in other scenarios. Reusability minimizes the duplication of components and also the implementation time.
Run-time Qualities	*Availability*	Availability defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period. Availability will be affected by system errors, infrastructure problems, malicious attacks, and system load.
	Interoperability	Interoperability is the ability of a system or different systems to operate successfully by communicating and exchanging information with other external systems written and run by external parties. An interoperable system makes it easier to exchange and reuse information internally as well as externally.
	Manageability	Manageability defines how easy it is for system administrators to manage the application, usually through sufficient and useful instrumentation exposed for use in monitoring systems and for debugging and performance tuning.
	Performance	Performance is an indication of the responsiveness of a system to execute any action within a given time interval. It can be measured in terms of latency or throughput. Latency is the time taken to respond to any event. Throughput is the number of events that take place within a given amount of time.
	Reliability	Reliability is the ability of a system to remain operational over time. Reliability is measured as the probability that a system will not fail to perform its intended functions over a specified time interval.
	Scalability	Scalability is ability of a system to either handle increases in load without impact on the performance of the system, or the ability to be readily enlarged.
	Security	Security is the capability of a system to prevent malicious or accidental actions outside of the designed usage, and to prevent disclosure or loss of information. A secure system aims to protect assets and prevent unauthorized modification of information.
System Qualities	*Supportability*	Supportability is the ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly.
	Testability	Testability is a measure of how easy it is to create test criteria for the system and its components, and to execute these tests in order to determine if the criteria are met. Good testability makes it more likely that faults in a system can be isolated in a timely and effective manner.
User Qualities	*Usability*	Usability defines how well the application meets the requirements of the user and consumer by being intuitive, easy to localize and globalize, providing good access for disabled users, and resulting in a good overall user experience.

Figure 2 – Software Quality Attributes