

Criticism of software engineering

From Wikipedia, the free encyclopedia.

Critics argue that many of the foundations of software engineering are inherently flawed. The following paragraphs list many criticisms and responses. Note that many of these criticisms apply to other human activities including business and education.

Contents

- 1 Managing Expectations
- 2 Poor Requirements
- 3 Rising Complexity
- 4 Ongoing Change
- 5 Ongoing Failure
- 6 Failure to Pinpoint Causes
- 7 Nothing New
- 8 Anyone Can Do SE
- 9 We Do Not Know What SE Is
- 10 No Software Science
- 11 No Mathematics
- 12 No Theorems About People and Projects
- 13 Meaning of Success
- 14 Magic
- 15 Software Creation is Inherently Creative
- 16 No Consensus

Managing Expectations

Criticism

One key to successful software engineering projects is managing the customer's expectations to something that can be built and delivered. So, the field resembles marketing or sociology more than traditional engineering with its responsibilities to society at large and the perils of legal liability when they fail to protect the public interest.

Response

Every profession manages expectations, including all branches of engineering. Moreover, *responsibility to society* means meeting the expectations of the general public, which is often a stakeholder.

Poor Requirements

Criticism

The requirements for most SE projects are incomplete or inconsistent. Some clients have little experience writing requirements. Other clients do not know what they want, and say "I'll know it when I see it" (IKIWISI). Even experienced clients who know exactly what they want may not precisely articulate their requirements. Clients often expect much more than they write in the requirements. And, requirement documents can describe applications that have no computable or practical solutions.

Response

One response is to avoid all projects with poor requirements, which would avoid most projects. Another response is using agile development and rapid prototyping to clarify project goals and deliver value (the most important requirements) quickly. Embedded systems are special in that they fit inside products that are designed by other engineers, who can sometimes define the software requirements completely.

Rising Complexity

Criticism

Critics argue that the complexity of requirements and user expectations have only increased. The probability of failure increases with the size, scope and complexity of the project. Technologies and practices have consistently improved over the years, but the gap between what is expected and what is delivered has not improved.

Response

Rising complexity actually shows the success of practitioners, because demand naturally follows supply. When clients demand more, it shows their belief that their demands will be supplied.

Ongoing Change

Criticism

Practitioners continually develop new technologies and practices and use them whenever possible. Critics argue that ongoing change proves that older technologies and practices were failures.

Response

Many view ongoing change as proof that software engineering successfully learns and grows. Traditional engineers also continually develop new technologies and practices.

Ongoing Failure

Criticism

Critics argue that incomplete or poorly designed systems are still too common. The early disasters in the field did not prevent subsequent disasters.

Response

No field that strives to do bigger and better projects has ever avoided all failures. Traditional engineers also have ongoing failures: automobiles kill 40,000 people every year in the U.S.; Three Mile Island, Chernobyl, and Bhopal Disaster harmed thousands; Space Shuttles Challenger and Columbia blew up; MTBE added to gasoline to reduce air pollution also contaminated drinking water. Although large, reliable software systems can be and have been constructed, software projects that fail during construction or in service are still too common. Ongoing failure is a problem for both traditional engineers and software engineers. Some claim that software engineering is already as predictable and reliable as many fields of engineering, such as space or biological engineering.

Failure to Pinpoint Causes

Criticism

Critics argue that unlike traditional engineers (who analyze failures, find precise causes, and set up guidelines to avoid them in the future), software engineers routinely fail to pinpoint causes of failure or delay precisely enough to avoid repeats in the future.

Response

Debugging is the activity of pinpointing the cause of failures in applications. Process improvement includes the activity of pinpointing the cause of process problems. Software engineers routinely pinpoint causes and then use the results to create better languages, databases, processes, and applications.

Nothing New

Criticism

Critics argue that software engineers created nothing on their own, but merely use what computer scientists already know.

Response

Software engineers developed optimizing compilers, make, cvs, extreme programming, scripting, and bug databases on their own, out of necessity. Regardless of who creates or improves technologies and practices, software engineers are bright enough to adopt the best ones.

Anyone Can Do SE

Criticism

Many bright people from other fields (engineers, scientists, business people) write spreadsheet templates or simulations, and eventually switch to writing large applications, and believe that they do software engineering. So, software engineering is not a special skill.

Response

Software engineering is a skill that is refined through long learning and practice. Software engineers are the ones who get the necessary education and experience, and keep up with evolving technologies, practices, and applications. This is true of every skill.

We Do Not Know What SE Is

Criticism

SE does not yield to the standard ways of categorization, under traditional definitions of engineering (calculus and science). This claim is often made by critics who want to impose their own definitions on everyone else.

Response

We know a lot about what software engineering is. Software engineering is grounded in SE technologies and practices, and applications; and in the community of SE practitioners. Of course, software engineers continue to disagree about many details.

No Software Science

Criticism

Civil, electrical, mechanical, and chemical engineering build on solid results from physics and chemistry. These results enable assembling complex systems in a principled and systematic way. No corresponding results are available for software: With software, we don't know how to systematically decompose complex systems into parts. Because there is no unified and agreed system for instruction on Software Engineering, it is difficult to evaluate the qualifications of those in the practice.

Response

Software engineering builds on solid results from computer science and information science. These results enable the building of very sophisticated software systems.

No Mathematics

Criticism

Many traditional engineering disciplines are based on logic and the mathematical disciplines of arithmetic, algebra, linear algebra and differential calculus, as well as the first principles of science such as Newton's laws, the laws of thermodynamics, Maxwell's equations, etc. Most of the engineering rules-of-thumb that make up

traditional engineering derive from the first principles of their respective physical and mathematical foundations. Software engineering has no such first principles that all practitioners agree to, and this makes it harder to justify it as an engineering discipline as opposed to a trade or a craft.

Response

Software engineers uses logic, proof theory, generating functions and wide variety of discrete mathematics when creating programs.

No Theorems About People and Projects

Criticism

No theorems explain why one software engineer is more productive than another. No theorems explain why some software projects succeed and others fail. Without such knowledge, *engineering* is impossible.

Response

Software engineering, like other engineering disciplines, is a complex social activity. No theorems explain why one mechanical engineer is more productive than another. No theorems explain why some civil projects go over budget and fail, for example why the Big Dig in Boston went way over budget, or why 2 space shuttles blew up.

Meaning of Success

Criticism

According to a study by the Standish Group (ref?) in 2000, 28 percent of software projects were complete successes (meaning they were executed on time and on budget), and 23% failed outright.

Response

Many engineering projects fail to live up to expectations: many bridges and buildings run over budget or schedule. Consider that 40% of all space shuttles have blown up and the rest have been out of service for years. Almost all custom housing projects run over budget and schedule. Success rates for software projects are meaningless without context.

Magic

Criticism

When programmers work hard and the system works, customers frequently do not appreciate how difficult the task was.

Response

This is true for every profession.

Software Creation is Inherently Creative

Criticism

Software is executable knowledge, which is discovered in a creative process, where trial and error, learning, and the ability to challenge one's assumptions are important. The true "construction" phase of software development is already automated by compilers and linkers. The difficult aspects relate to gathering requirements and designing systems. These are more like a craft than a science or engineering task. So, the potential benefit of software engineering is limited to making it easier for developers to try out ideas, to discover errors earlier, and to give them information about the state of a software system.

Response

Many software engineers use agile software development to embody the creative nature of software engineering and to foster learning. The area of requirements engineering looks at how to elicit and capture what a system

should do and how this knowledge can be used in software engineering activities.

No Consensus

Criticism

In traditional engineering there is a clear consensus how things should be built, which standards should be followed and which risks must be taken care of: If an engineer does not follow these practices and something fails he gets sued. There is no such consensus in software engineering: Everyone promotes their own methods, claiming huge benefits in productivity, usually not backed up by any scientific, unbiased evidence.

Response

SE is a young discipline. As consensus emerges, SE will be thought of as a mature discipline.

Retrieved from "http://en.wikipedia.org/wiki/Criticism_of_software_engineering"

Categories: Software engineering

-
- This page was last modified 18 April 2005 00:58.
 - All text is available under the terms of the GNU Free Documentation License (see **Copyrights** for details).