**History Of Software Engineering**

Software engineering has evolved steadily from its founding days in the 1940s until today in the 2000s. Applications have evolved continuously. The ongoing goal to improve technologies and practices, seeks to improve the productivity of practitioners and the quality of applications to users.

1945 to 1965: The Origins

The term software engineering first was used in the late 1950s and early 1960s. Programmers have always known about civil, electrical and computer engineering and debated what engineering might mean for software.

The NATO Science Committee sponsored two conferences on software engineering in 1968 and 1969, which gave the field its initial boost. Many believe these conferences marked the official start of the profession.

1965 to 1985: The Software Crisis

Software engineering was spurred by the so-called software crisis of the 1960s, 1970s and 1980s, which identified many of the problems of software development. Many software projects ran over budget and schedule. Some projects caused property damage. A few projects caused loss of life. Some used the term software crisis to refer to their inability to hire qualified programmers. The software crisis was originally defined in terms of productivity, but evolved to emphasize quality.

Cost and Budget Overruns: The OS/360 operating system was a classic example. This decade-long project from the 1960s and 1970s eventually produced one of the most complex software systems ever created. The OS/360 was one of the first large, 1000 programmer, software projects. Fred Brooks claims in 'Mythical Man Month' that he made a multi-million-dollar mistake by not developing a coherent architecture before starting development. Property Damage: Software defects can cause property damage. Poor software security allows hackers to steal identities, costing time, money, and reputations. An expensive European Ariane rocket exploded because of software.

Life and Death: Software defects can kill. Some embedded systems used in radiotherapy machines failed so catastrophically that they administered lethal doses of radiation to patients. Peter G. Neumann keeps a contemporary list of software problems and disasters at Computer Risks.

The software crisis has been slowly fizzling out, because it is unrealistic to remain in crisis mode for more than twenty years. SEs are accepting that the problems of SE are truly difficult and only hard work over many decades can solve them.

1985 to Present: No Silver Bullet

For decades, solving the software crisis was paramount to researchers. Seemingly, they trumpeted every new technology and practice from the 1970s to the 1990s as a silver bullet to solve the software crisis.

Tools, discipline, formal methods, process, and professionalism were touted as silver bullets. Tools: Especially emphasized were tools. Structured programming, object-oriented programming, CASE tools, Ada, documentation, standards, and UML were touted as silver bullets. Discipline: Some pundits argued that the software crisis was due to the lack of discipline of programmers. Formal methods: Some believed that if formal engineering methodologies would be applied to software development, then production of software would become as predictable an industry as other branches of engineering. They advocated proving all programs correct. Process: Many advocated processes and methodologies like CMM. Professionalism: This led to work on a code of ethics, licenses and professionalism.

In 1987, Fred Brooks published the 'No Silver Bullet' article, arguing that no individual technology or practice would ever make a 10-fold improvement in productivity within ten years. Debate about silver bullets raged over the following decade. Advocates for Ada, components, and processes continued arguing for years that their favorite technology would be a silver bullet. Skeptics disagreed. Eventually, almost everyone accepted that no silver bullet would ever be found. Yet, claims about silver bullets pop up now and again, even today.

Some interpret 'no silver bullet' to mean that SE failed. The search for a single key to success never worked. All known technologies and practices have only made incremental improvements to productivity and quality. Yet, there are no silver bullets for any other profession, either. Others interpret no silver bullet as proof that SE has finally matured and recognized that projects succeed due to hard work.

**Major Developments**

There are a number of areas where the evolution of software engineering is notable. Emergence as a profession: From the mid-1990s to the mid-2000s, software engineering emerged as a bona fide profession, to stand beside computer science and traditional engineering.

Role of women: In the 1940s, 1950s, and 1960s, software was often written by women. Men often filled the highest prestige hardware engineering roles. Grace Hopper and many other unsung women filled many programming jobs during the first several decades of software engineering. Today, fewer women work in software engineering than in other professions. Saying that this is sexual discrimination is too simple because it related directly to individual identity. In one sense, software engineering is the masculinization of programming. The roles women fill in SE continue evolving. Today, more women in software engineering fill the social roles of analysis, training, documentation and management; and fewer do hardcore technical development.

Processes and Methodology: Processes and methodologies have become big parts of software engineering. Many practitioners resist process, which often treats them impersonally like machines, rather than like people.

Cost of hardware: The relative cost of software versus hardware has changed substantially over the last fifty years. When mainframes were expensive and required large support staffs, software projects could be expensive. Because powerful PCs are cheap, software projects must become cheaper in comparison.