**NO SILVER BULLET**

**Abstract -** All software development turns around essential and accidental tasks. There was a significative increase in software productivity due to overcoming the artificial barriers that made accidental task immeasurably hard.

**Introduction -** Werewolves are mythical beasts that would remain normal until transforming unexpectedly in a terrible monster. The only way to destroy these monsters it’s killing them whit a silver bullet. We can compare a software project to a werewolf in the sense that it may become a chimera hard to bring down. What we are looking for to be this silver bullet is something that can reduce the software cost as fast as the hardware cost have been reduced.

**Does It Have To Be Hard? - Essential Difficulties**

The big anomaly is that hardware progress rate is too fast when compared with software progress. According to the author, the difficulties that handicap the software progress can by divided into **essence** (the problems inherent in the nature of the software) and **accidents** (not inherent difficulties related to software production). The author presents the following properties inherent to the irreducible essence of modern software systems:

**Complexity –** Because the complexity of a software entity increases in a non-linear fashion compared with its size, software is highly complex, creating technical and managing problems, requiring knowledge to lead to viable products. It originates communication difficulties which results in product flaws, cost overruns and schedule delays. It also creates enumeration issues and lessens the comprehension of the software.

**Conformity -** This problem is based in the conformity facing other interfaces, needing to be in conformity with existing systems. It happens because the software is newer and its easier to edit when compared to older already existing systems.

**Changeability –** Every software entity is under constant pressure for change. The software part that most feels this pressure it’s its function, which must be easily manipulated to be extended to new purposes. Good software doesn’t have problems supporting new types of hardware that arrive to the market.

**Invisibility -** Software is constituted by diverse directed graphs, turning it invisible and not visualizable which makes it very abstract, withdrawing the power of conceptual tools.

**Past Breakthroughs Solved Accidental Difficulties**

Taking a look to the past steps in software technology that have been really positive we can conclude that they were **accidental** difficulties, such as **high-level languages, time sharing** and **unified programming environments.** In the first one, the author makes us know that the progressive use of high-level programming languages has been the most powerful stroke for software productivity, reability and simplicity. We can divide the program in the abstract and concrete parts. High-level languages offered many constructs to the programmer for the abstract program. In its turn, **time-sharing** was an huge improvement not only in programmers productivity but also in the products quality, preserving immediacy and a good overview of complexity, decreasing system response time. The principle idiosyncrasy of **unified programming environments** is that they attack the accidental difficulties of using programs together.

**Hopes For The Silver**

Considering the technical developments that are considered silver bullets we begin with **Ada and other high-level languages,** which started abstract types of data and hierarchical structuring, modularizing data and improving design concepts. **Object-oriented Programming** allowed the developers to express they design without information content or syntactic material with data hiding based in abstract data types and classes. **Artificial Intelligence b**rought extreme gains on the productivity and quality of the software, using expert systems technology. The use of **Expert Systems –** programs that contain a generalized inference engine and a rule base – gave new conclusions explaining the obtained results, bringing innumerable advantages. Also, one of the most powerful technique with many known solution methods and extensive analysis that puts few parameters in problems and is used in sorting programs is **“Automatic” Programming.**

We can also believe that **Graphical Programming, Environment and Tools, Program Verification** and **Workstation** are great silver bullets that helped overcome obstacles in software development.

**Promising Attacks On The Conceptual Essence**

One of the most promising attacks on the conceptual essence is **Buy Versus Build.** This attacks tells us that the most radical possible solution to apply to software construction is not to construct at all, making clear that products are cheaper to buy rather to build and that bought products have tendency to be much better documented than homegrown software.

**Incremental development-grow, not build, software** reveals that the secret is that software is grown, not built, turning backtracking easier, making the software to lend itself to early prototypes.

In our opinion, the most important attack that the author presents are **Great Designers,** because good designers follow good practices that can be taught. The very best designers produce much better structures with less effort. Developing great designers, we develop greater software.