**Abstract:**

When talking about Software Construction we must not avoid two essential tasks:

* (1) Modelling of conceptual structures that make the abstract software entity;
* Accidental tasks, which is the representation of the abstract entities in programming language.

Now, unlike in the past, engineers put almost null effort into overcoming constrains that would reduce the time wasted with accidental activities. Then, nowadays it is more important to be concerned with the (1).

**Introduction:**

Today, we fear all the problems that may appear upon us just as much as werewolves are feared in tales. But our silver bullet, which is, in this case, something that allows us to reduce the software production costs, is non-existent and will highly likely not appear in a near future. This happens since it is not probable to appear a lonely development that will improve our technology dramatically. However, constant and consistent small improvements will help us develop.

productivity

**Does it have to be hard? – Essential Difficulties**

We cannot expect software progress to be as fast as hardware’s, which has to do with the nowadays development on human’s technology itself. This said, we must take into consideration both intrinsic difficulties of software – essence problems - and the ones related with software’s production – accidents. So, we shall start with essence obstacles.

These essence problems are completely abstract. Nonetheless, they are just as much exact as abstract. So, we must consider four properties that make the essence irreducible:

1. Complexity;
2. Conformity;
3. Changeability;
4. Invisibility.

**Complexity**

The software complexity is quite higher when compared to that of a physical construct, since we can’t really find two separate sections that look alike. This means that the ideas behind on how to conceive software are also, in themselves, more complex. Besides that, the improvement of a software involves not only the increase of the code’s size, but also the complexity of it, which increases the difficulty of the coding part.

This complexity is the cause of several problems, such as:

* Lack of team communication;
* Unreliability of program, due to all possible states of the program;
* Programs of hard use;
* Hard to increase new functions without side effects;
* Less security.

Besides that, it also escalates problems related with the conception of the software, with how one can control and manage the whole project and the learning of the code.

**Conformity**

Unlike, for example, physics, who also deal with really complex systems, software engineers must not have faith that there are principles and laws yet to be found that were designed by some superior entity. The last ones deal with completely random complexity, forced by the human nature itself.

**Changeability**

Just as physical stuff like houses or transport means, software also has to keep being updated, which means to be changed. But, due to its never ending agility to be changed, the updating process never stops. Besides that, the fact that the users keep trying to maximize the uses of a program, operating it even to cases that it was not designed and the fact that new vehicles of software transport keep appearing, this one must be updated.

**Invisibility**

Unlike, for example, building, software cannot be seen. This makes it hard for us to capture clear contradictions. This also means that when trying to diagram the software structure, several graphs representing relationships must be considered, which makes hierarchies difficult to visualize, leading to a lower conception of conceptual tasks.

**Past Breakthroughs that Solved Accidental Difficulties**

1. High Level Languages
2. Time-sharing
3. Unified Programming Environments

**1**

The use of these high level languages has clearly simplified software by reducing some of its complexity since we do not need to work with lower levels of computing

**2**

**3**

The use of standard formats allows us to apply any new tool to almost every program. This also makes it possible for structures to communicate with others more easily and in a more efficient way.

**Hopes for the Silver**

**Ada + high-level language advances**

Ada is a language whose philosophy is quite more advanced that the language itself. It matches the evolution in language concepts. It is based on modules, abstract types and hierarchies.

**Object Oriented Programming**

A large community believes that OOP may be the solution to the lack of software productivity problem. But this will surely not happen until the irrelevant obstructions of type specification remain.

**Artificial Intelligence**

Nowadays, AI is used mainly for speech/image recognition, and this specificity makes it unable to become really important in the programming practice

**Expert Systems**

**“Automatic” programming**

Is seen like a euphemism for programming with a higher-level language than the current ones. Basically it is the creation of a program to solve the problems by taking access to the parameters and then choosing from a library of solution methods

**Graphical Programming**

Although flow charts may look like a pretty good idea to represent the global overview of the software structure, they are basically useless taking into account the complexity of them

**Program verification**

Although this is a very important subject, the verifications mean a really big work and do not even assure error-proof programs

**Environments and tools**

Nowadays there is not much room for improvements in what relates with this

**Workstations**

We can in fact use more powerful workstations, but the truth is that there would be no significant enhancements in our programming capacities.