STORY OF A WAR AGAINST SOFTWARE COMPLEXITY

By Ir J.A.J. van Leunen

Retired physicist & software researcher

Location: Asten, the Netherlands

Website: www.scitech.nl

Communicate your comments to info at that site.

ABSTRACT

This is the account of the course of a project that had the aim to improve the efficiency of embedded software generation with several orders of magnitude. All factors that determined the success of the project are treated honestly and in detail.

CONTENTS

PRELUDE

In 1995 the physicist HvL was employee of an internal software house of a large electronics company. His speciality was the creation of scientific software. Then he got the invitation from a software strategist HdV to join the semiconductor department in order to resolve a quickly emerging problem. The costs of complex embedded software were growing exponentially and this would cause severe problems in the next future. The reasons why the costs of software generation grow exponentially are the growing size and the growing complexity of the embedded software that goes into high-tech appliances. One of the reasons of the non-linear growth of costs is the growth of complexity. But the exponential growth of costs is mainly caused by surpassing of the available resources, which on its turn required measures against expected internal and external damage claims. In many cases software projects were stopped when the costs were expected to explode, or when they did not seem to reach the expected result.

ANALYSIS

There exist several possible solutions to this dilemma. One is to move the software development to low wage countries. Another is to apply open source software. A third possibility is to increase the quality of the software generation process. One way to do this is to improve the control of the flow of the generation process. Another way is to improve the way that software is generated. The management of the electronics company tried all these possibilities. Improving the control of the flow of the software generation process has little sense when the generation process itself has severe defects.

SETTING

The electronics company is successful in the generation of hardware. This is for a large part due to the fact that hardware is generated via a modular approach. This is one of the reasons that the research lab of the firm created a dedicated way to create modular embedded software. However, this is a rather closed system and it is directed to the direct need, the generation of software for consumer appliances. Still it is intuitively felt that a modular approach will improve the effectiveness of software generation. There are also many objective reasons for this point of view.

HISTORY

The electronics firm was not very successful with its software projects. Many software projects were stopped after having burned hundreds of man years and millions of dollars, leaving the project leaders, system architects and the software designers back in despair. For that reason relief was sought in outsourcing of the software generation. One form of it is the use of open source software. Parallel to it the internal software generation was moved for a significant part to low wage countries like India. This was only a short time solution. The exponential growth of costs took its toll there as well.

Also the switch to open source software was no smart decision. The electronics firm had no control over the way that the open source software was evolving and the open source software generation suffered the same bad habits as the present-day commercial software generation process does. Commercial software generation and open source software generation are both non-modular. There exists no healthy and lively software modules market that stimulates the diversity, availability, accessibility and favourable quality/price ratio that characterizes the hardware modules market. Thus, the high-tech hardware appliances industry is still confronted with the negative aspects of the current software generation technology. It drives their costs high and the fragility of the software is transferred to the hardware products that include the software. The ineffectiveness of the software generation affects the affordability and the time to market of the hardware products.

STRATEGY

A small group of experts consisting of the software specialist HvL, the software strategist HdV and a software marketing specialist WR studied the resulting possibilities and concluded that a drastic change in the way that software is generated is a promising solution of the problem. The way that hardware is generated was taken as an example. Hardware is generated mostly in a modular way. Modularization reduces the relational complexity of the design and construction process. It also enables partition and delegation of the design and construction work. It even enables a flourishing modules market.

APPROACH

The group tried to interest vendors of embedded software generation tools to join the enterprise. It was obvious that international standards would play a crucial role. So, the group stimulated the management of the electronics firm to involve other electronics firms and the OMG. All these measures lacked sufficient success. The tool vendors were interested, but used the opportunity to monitor whether their current way of operation was endangered. They did not really take part in the development. The other electronics companies took the role of an observer and asked for a convincing demo of the concept. OMG lets standards create by the interested parties. It is not usage that the standard is introduced by a single company.

WHAT HAPPENED

The group encountered severe resistance against their intentions from internal software development groups, because it was expected that the generation of the modules would be outsourced to the suppliers of the software modules market. This fear is realistic. On the other hand it became more and more clear that the internal software generation capabilities were not measured up against the task to create large and complex embedded software systems. Several costly debacles proved this. Especially managers, including the managers of software groups, showed that they lacked a proper feeling for the factors that influence complex software generation.

ATTACK

The group decided to create a demo version of the modular software generation system that included major parts of the envisioned system. This includes software module development tools, system configuration tools, web and local file based repositories that act as searchable exchange places for machine and humanly readable specifications of modules and interfaces and central services that act as a marketplace for software modules. The module development tool can generate skeleton modules and it can generate the interface definitions from specifications that are retrieved from web based or local repositories. The tool helps filling the skeletons with working code. The configuration tool retrieves specifications of modules and interfaces from the repositories. It can retrieve the binaries of modules from the market place or from a local store. It enables the mostly automatic assembly of modules into target systems. It adds a dedicated RTOS that consists of automatically generated modules. The RTOS provides automatic memory garbage collection. The central service collects specifications from the module developers and distributes these to the web based repositories. The central service also collect the binaries of the modules and stores the specifications and the binaries in its banks. The central service acts as a modules market.

SET-BACK

The project of the group was severely hampered by the dot com crisis in 2001. This stopped all long term research projects and brought the funding of the group to a minimum. The development of the demo continued at a low pace and stopped in 2004. At that time most planned parts of the demo worked at least for a large part.

REMNANTS

The central service worked partly. The development tools are functioning. Modules can be generated and the configuration tool can assemble systems from these modules and add a service layer that consists of automatically generated dedicated modules. The service layer includes garbage collection. It uses connection schemes and scheduling schemes that dynamically control the switch between system modes. The created system does not contain a HAL and it does not contain interrupt services. In stead it relies on the services of a virtual machine or a POSIX OS. This is not the target to work on top of hardware but it is good enough for most demonstration purposes. The tools generate software in C++, but as a bonus it can

deliver C# code. That code works on top of a dotNet virtual machine. The tools and central services are written in C#.

Apart from SW/SW interfaces the modules may contain HW/SW interfaces. Streaming interfaces and the notification interfaces that handle interrupts were planned. The skeleton of the modules are modelled after Microsoft's Component Object Model (COM), but the IUnknown interface is replaced by the IAccessor interface. That interface replaces the AddRef and Release functions with a ResetInstance routine. In stead of the designer, the system is made responsible for the garbage collection. For that reason the new module skeleton is named Robust Component Object Model (RCOM).

GOAL

The demo was planned to demonstrate the generation of real-time embedded software. That goal is not reached. However, many aspects of the planned target are shown in the completed part of the demo. That part offers trust in the feasibility of the final goal.

LESSONS

The project also learned many valuable lessons.

- The current suppliers of software generation tools are not interested in a drastic change in the way that software is generated.

- Despite the fact that embedded software is causing major problems, the companies that produce high-tech appliances or high-tech systems are hesitating to cooperate in improving the software generation process. Software generation is not their strength.

- This world is not good in organizing actions that are rather complex. For that reason it is difficult to arrange standards on new subjects.

- It is difficult to motivate management to enter new inroads when the reasons are not very simple and require insight in the topic.

- Managers of these days are interested in short term low risk solutions. They are not interested in long term solutions even when they promise high profits.

- The same holds for today's investors.

- Although most involved people intuitively see that a modular approach provides a better effective generation process and easier support management, most of these people forget that without a suitable modules market the modules are too expensive and too scarce to make the assumption true.

- A modular system generation approach has no sense when it does not include an integrated and well functioning modules market. This also means that a system of web based and local repositories that contain the specifications of modules and specifications must be involved as well.

- Given enough resources, even a tiny group of determined software experts can design and construct a working version of a modular software generation system that includes all essential parts.

CONCLUSIONS

This world is not good at creating new standards. However, we are good in accepting default standards. Large electronic firms seem incapable of creating a suitable software generation system. Understandably, the existing software industry appears not willing to give up the profits that they retrieve from the current deplorable way of software generation.

WAY OUT

There still exists a possible way to get out of this misery. When a small group of enthusiastic software developers and venture capital investors start with a project that establishes a working version of a modular software generation system that includes all ingredients to get a successful result, then they may cause the seed that will extend like an oil drop and smother the current way of software generation.

In a world where such a system exists the complex software assemblies are no longer created by a genial system architect and hundreds of man years of expensive programmers but instead by a creative modular system assembler that uses automated tools to construct his target in a fraction of the time, with a fraction of the resources and with a fraction of the costs compared to his present-day colleague. He retrieves his modules from a modules market and he may also design and produce some missing modules. In a later phase he may decide to offer these new modules on the market.

His present-day colleague produces software systems, whose structure resembles a layered set of patchwork blankets. Even the most ingenious architect cannot oversee the details of this complex architecture. Therefore the system cannot be completely described properly. Thus, it cannot be tested fully and nobody can guarantee its proper functioning. Modular systems are inherently less complex. Especially its system configuration is orders of magnitude less complex. This results in a better manageability of the complexity and a higher robustness. On its turn it results in a better chance to be able to guarantee its proper functioning.

DISCUSSION

The modules market is very democratic. Everybody that owns an appropriate modules development system can participate and fill a niche of the modules market. The modules market is a good replacement of the market for open source software. It has the advantage that the module developers can earn money for the intellectual property that they invested in the design and construction of the module. Still the products stay very affordable. In contrast the open source software community is non-democratic. In many cases the community forbids the contributing software developers to earn money from their intellectual property investments.