The Problem with DevOps

Otto Linnemann[1]

[1] *https://github.com/linneman*

March 2020

Exeriences with the introduction of DevOps Methodologies introduced in order to provide and maintain complex build environments in the domain of the development of mobile communication equipment for the automotive industry are presented. These were implemented at the same time as other measures due to organisational changes in the context of a company acquisition. The report reflects the experience from the perspective of a software developer.

1 Objective

This report describes the experiences with the introduction of DevOps methodologies in the development of complex software systems for mobile communication and the automatic emergency call for automotive applications. This technology is often referred to as 'connectivity' in the context of the automotive industry and is one of the four megatrends that have shaped the automotive industry since the beginning of the 21st century [4].

This area is therefore currently receiving a great deal of attention from market participants. In the development period reported in the following, the medium-sized company in which the projects were started was taken over by an internationally operating group, which led to extensive restructuring within the organisation. The official introduction of DevOps methods also took place during this period.

Before we come to the actual topic of DevOps, we must briefly outline this technology area. Only then is it possible to classify the specific requirements and problems.

2 Application Domain

With the introduction of the mobile phone, the integration of this technology into cars was practically simultaneous. This development essentially took place in three phases. Initially, the mobile terminals were connected to the vehicle electronics by means of a charging cradle. Subsequently, the introduction of Bluetooth enabled a wireless connection between the vehicle and the terminal device and created a manufacturer-independent standard for the first time. In these systems, the antenna of the mobile end device is used, which can be problematic in modern LTE mobile networks when using shorter wavelengths. For this reason, vehicle manufacturers are integrating complete mobile radio modules into the vehicle, which makes new applications such as software updates via broadband connections and

electronic emergency calls possible in the first place. Since April 2018, the use of such a system is mandatory for the type approval of new vehicles in the European Union which explains the increased market interest in this technology. In the following we cover exclusively the latter systems. So the focus is on the development of software for mobile data modems for use in automotive applications.

As the iPhone and smartphones in general became more and more widespread, it became increasingly common to separate the technology of the end devices from the actual mobile phone technology. Today, both chipsets and complete data modems are available as individual components in hardware and software. These always contain several processor cores. The newer LTE systems from market leader Qualcomm also include a processor core reserved for customer applications, typically running a Unix-like operating system. In the projects presented here, the application processor core is based on GNU/Linux.

Most software components are already supplied by the modem supplier. However, their use in automobiles requires modifications to almost all components. In addition, a customer-specific CAN controller is used to connect to the vehicle network. The resulting software system is therefore very complex.

3 Development Teams

In the environment described here, software developers in the automotive domain often have experience exclusively in classic embedded development. Many are graduate electrical engineers and therefore have no formal education in computer science. In many cases, only basic knowledge of the Unix/Linux operating system and the libraries, frameworks and tools used therein is available. Nevertheless, we have been able to make the experience that a number of developers were able to familiarize themselves with the topic surprisingly quickly if they were interested. We could observe that those developers who showed an interest in these technologies were moving to the software teams that were responsible for these developments. In smaller companies such changes are usually much easier, as the formal organisational structures are usually not so distinctive. An open corporate culture also enabled the exchange of knowledge within and between the working groups.

4 Starting Point

At the very start of the first project, the following procedures were largely implemented independently within the team responsible for LTE modem development:

- Setting up a knowledge database in the form of a Wiki
- Use of Git [2] to version all source code
- Use of Gerrit [1] to review added changes in the existing source code
- Provision of software releases via a manifest file based on Android/Repo [5]
- Provision of formal documentation of the required steps to release a new software version
- Development or adaptation of standard API's for abstraction of manufacturer-specific modem functions

On this basis, the first customer projects were initiated by a second team. Both teams were located closely to each other, which was one of the reasons why problems in general could be solved quickly.

However, difficulties were immediately apparent which could not easily be overcome over a longer period:

- Due to the complexity of Git, releases could only be created by a few team members.
- As a result, the integration of software changes was delayed, sometimes considerably
- Attempting to address this problem by introducing an automated build system (Jenkins) failed because it could not detect dependencies between subprojects

and the runtime for a complete build on the hardware used at the time was in the range of a few hours.
- The integration of the delayed platform releases into the customer project caused additional difficulties and led to even greater delays
- This resulted in considerable effort regarding the analysis of error reports from the field, which had been corrected for a while already but the corresponding changes had not yet been integrated.
- Developers of both teams were not able to easily analyse system behaviour because the platform and the customer project used a different source code management, build system and hardware
- External developers could not easily be included, as parts of the source code could not be passed on by suppliers due to existing license conditions.

5 Introduction of DevOps

The acquisition of the division was directly linked to the goal of expanding the business. Therefore a new Head of Development was recruited for the software department. One of his first measures was the establishment of a new department for Development Operations (DevOps) which was staffed exclusively through external employees with temporary contracts. The primary tasks of this department were:

- Establishing an infrastructure for software builds in the cloud
- Provision for a standardised and virtual development environment (SDK) based on Docker containers hosted in this cloud
- Continues Integration of all changes enforcing automated builds, code quality checks, verification and testing

The new management saw these measures as the key to introducing further far-reaching changes:

- Development of a standardised application toolkit instead of a monolithic customer application to improve code reuse
- Involvement of external development sites to scale up the work force and for cost reduction purposes
- Introduction of a much stricter development process to improve the control of external and internal work
- Introduction of metrics to assess code quality and performance of employees and suppliers
- Meeting customer requirements for participation in software development

In addition, the integration into the Group made it necessary to adapt existing processes and systems. The corporate IT department officially does not allow the use of Linux systems specified by the system supplier. This problem was hoped to be solved by the cloud infrastructure. However, since this is accessed via the corporate network, the requirement could not really be met. The problem regarding the transfer of source code protected by licenses could also not be solved by switching to the cloud.

However, the most serious limitation proved to be the fact that the management of the cloud infrastructure was carried out by a different corporate department. The actual advantage of a cloud solution, i.e. to adapt the computing power provided to the demand, could not be used. As a result, the cloud solution proved to be extremely expensive and inefficient. Software developers were confronted with sometimes considerable latencies in the range of seconds in dialog mode with the new virtualized build systems in a simple text console. The use of the meanwhile common Integrated Development Environments (IDE's) was mostly not possible. It is therefore not surprising that the newly built cloud system was widely rejected in the development department.

This situation also proved to be extremely frustrating for the DevOps team. They quickly realized that under the given circumstances it would not be possible to meet the require-

ments of the developers. It was therefore more or less accepted that developers continued to use their legacy systems. But since the management officially insisted on using the cloud, at least official releases had to be built in the new cloud infrastructure.

Since most of the developers refused or were not able to work with the cloud system at all, the creation of new software releases was quickly taken over by the DevOps department. Due to the associated expert knowledge, individual DevOPs employees could quickly become indispensable.

This situation was considerably aggravated by the atmosphere of competition between permanent and freelance employees and external companies. As a result, key qualifications were only passed on to a very limited extent. Developers were no longer able to build software releases themselves or to follow the integration process. The DevOps software integrators do not have the necessary domain knowledge and are not able to test the generated software thoroughly. For this reason, software was sometimes delivered to customers untested. This practice still continues today.

6 Lessons Learned

The entire transformation process described here was accompanied by keywords such as continous integration, agile development and the use of a purely cloud based infrastructure. None of this could be achieved in the original sense. One of the core ideas of agile software development is to overcome the divide between management and developers [6]. As already mentioned, the exact contrary has happened. It is therefore obviously advisable to be cautious when implementing the methodologies that are currently in vogue.

We see a fundamental problem in the overspecialization in software development where developers, software integrators, DevOp's specialists work in different departments and working groups which tend to use different tools. In the case described here, we were able to observe that DevOps specialists acted as super software integrators. The fact that this group alone had an overview of the integration and release process at the end of the day was particularly fatal after such key persons left the company.

In earlier projects, it was common practice to document the steps for creating a new release candidate in a check list and releases were created in succession by each developer. Such dependencies on individuals can thus be avoided very effectively.

A mixture of DevOps and development tools should be avoided in any case. The actual build process is the task of the build tools. It should actually be self-evident that all developers use at least the same build tools. In the projects shown here this was not the case, which led to countless problems and misunderstandings.

The file system and bootimages are generated in the given project using the Yocto/OpenEmbedded [7] Meta Buildsystems. This system can now be considered the de facto standard in the field of embedded Linux applications. Its handling, however, was regarded as too complicated for external developers. Since for various reasons a SDK could initially not provided, these developers created their own build environment with different make system which created further difficulties during integration. Furthermore, instead of using the powerful tools which Yocto provides we could often observe that Yocto's tools have been replicated using external shell and Jenkins [3] scripts. We consider it very problematic that many of these scripts are not managed as integral part of the projects source code. This makes it difficult to trace how a particular release was actually created.

The continuous integration of any change has proven to be impractical due to long compile times and hard to capture cross repo dependencies between different subprojects. However, we have had good experience with a nightly build and release system tailored to our requirements which exclusively provides remotely executed build commands without

any further complex logic encapsulated. It is only assumed that a complete set of changes is integrated into the development branches used at office hours. The following morning all developers will be informed about the build results by email. If the build has run without errors, a new release candidate is automatically created and its version number and the list of all changes contained in it are distributed by email and via the integration server website. The time-consuming and repetitive work of manually creating release candidates is no longer required in this case.

References

[1] Gerrit Code Review `https://www.gerritcodereview.com`

[2] Git, a distributed version control system `https://git-scm.com/`

[3] Jenkins, a self-contained, open source automation server `https://jenkins.io`

[4] Disruptive trends that will transform the auto industry `https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/disruptive-trends-that-will-transform-the-auto-industry`

[5] repo - The Multiple Git Repository Tool `https://gerrit.googlesource.com/git-repo`

[6] Robert C. Martin - The Land that Scrum Forgot `https://www.youtube.com/watch?v=hG4LH6P8Syk`

[7] Yocto Project, an open source collaboration project that helps developers create custom Linux-based systems `https://www.yoctoproject.org`