Minimally Invasive Migration to Software Product Lines

Hans Peter Jepsen, Jan Gaardsted Dall
Danfoss Drives
{hans_peter_jepsen ,gaardsted}@danfoss.com

Danilo Beuche
pure-systems GmbH
danilo.beuche@pure-systems.com

Abstract

Danfoss Drives - one of the largest producers of frequency converters in the world - is in a situation like many others: it has to produce a number of product series with an increasing number of variants, while at the same time decreasing time-to-market and keeping development costs low. As part of the strategy to master this challenge, Danfoss Drives decided to reduce software development efforts by migrating to a product line approach.

This paper describes the approach Danfoss Drives took to successfully introduce Software Product Line Principles into its organization.

1. Introduction

Danfoss Drives - one of the largest producers of frequency converters in the world - is in a situation like many others: it has to produce a number of product series with an increasing number of variants, while at the same time decreasing time-to-market and keeping development costs low. Several important steps have been taken over time in order to meet these challenges. Products are being developed on a common product family software architecture based on object-oriented principles. This architecture serves as a foundation for systematic software reuse. However, a product family architecture is not enough. The latest step is to structure the development process along the principles of software product line engineering.

This article will describe the migration of the Danfoss Drives software development towards a product line approach and discuss some findings of use to others starting a similar migration process.

Before going into the details of this migration, a brief introduction of the application domain (frequency converters) and Danfoss Drives' Development organization setup will be given. Section 2 briefly reviews the history of Software Development at Danfoss Drives, since it is important to see the steps in order to understand the result. The following section 3 discusses the set-up phase of the migration where most of the "thinking" was done and also describes how the initial software family was constructed from existing software systems. Section 4 will then describe the rollout phase where the different project teams migrated their development processes. The following section 5 summarizes the initial learning-points from the rollout. We then take a brief look at tooling issues and conclude with a summary and some remarks about the future of product line development at Danfoss Drives.

1.1. The Domain

A frequency converter is an electronic power conversion device used to control shaft speed or torque of a three-phase induction motor to match the needs of a given application. This is done by controlling the frequency and corresponding voltage on the (three) motor cables. Applications for frequency converters are found in many domains. For instance using a frequency converter in connection with a fan to maintain a desired room temperature while conserving as much energy as possible (see Figure 1), other examples are moving rollers in plants or pumping water using electrical pumps.

This wide range of possible applications requires a high degree of variability in the product line. The parameters of the motors to be controlled vary. The environments in which the converters have to be integrated are very diverse, in some cases they work standalone, some are connected using (different) field busses and protocols, the integrated control algorithms vary, etc.

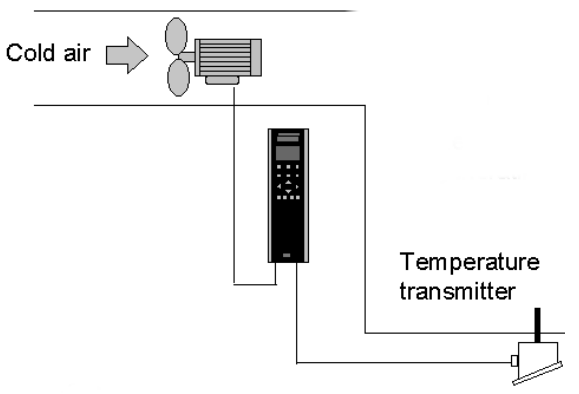

Figure 1 Frequency Converter Application Scenario: Fan control

Due to hardware and cost constraints, among other reasons, it is not possible to equip all frequency converters with universal software fit for any imaginable purpose.

Danfoss Drives has traditionally released three main frequency converter series – for industry, for HVAC (Heat, Ventilation and Air-conditioning) and for the water segment – and many special products (e.g. for crane and specific textile applications) as well as some brand labelled products.

Most of these products are very flexible and have several hundred configuration items (called parameters), that give the customer the possibility to adapt the frequency converter to the application. Furthermore the customer has the possibility of adding optional modules that enable field bus communication (e.g. Profibus) and extended I/O.

1.2. The Embedded System

A frequency converter from the new high end product series, where the first products were released in 2003, is a distributed system with at least three microprocessors. The main processor is an ARM7 running several hundred thousand lines of C++ code. This is tightly coupled with a powerful DSP with peripherals selected for motor control. These two processors are located on a control board, which is used in all products with minor variants.

The third microprocessor is located on the power unit, which exists in many different sizes depending on the size of the motor is has to control. This processor is a small 8 bit processor.

The optional modules – a graphical control and configuration device and up to four modules from a broad palette of communication modules and I/O extensions – are all equipped with one or two microprocessors. The media for communication between these processors are high speed CAN, SPI and asynchronous serial.

The software architecture has three main parts (see Figure 2).

One is the continuous processing part, which is time-driven at a very high rate. The input part of this samples or receives set points and/or feedbacks and makes calculation on these. The output part is a very advanced motor control, where one of several control algorithms controls the motor.

Next is the event-controlled part, where the input side (the CommandHandler) receives requests (e.g. start or stop) from digital I/O or field busses, decides what the resulting command is and sends it to the output part (the MotorManager). Here a state machine is responsible for selection, activation and deactivation of the different control algorithms.

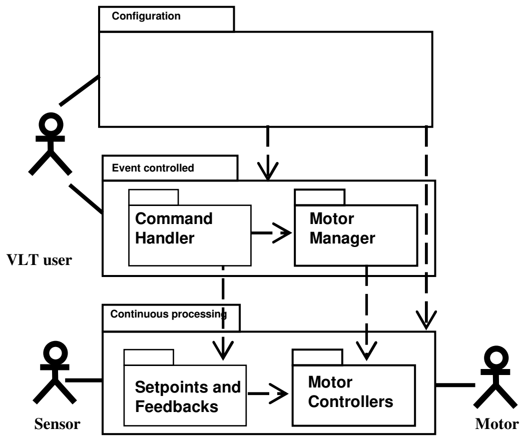

Figure 2 Architectural breakdown of a frequency converter

Finally there is the configuration part, which receives user–specified parameters, validates these and sends them to subscribing subsystems in the other two parts.

The continuous processing part and the event controlled part and the relation between these are described in detail in [1].

1.3. The Development Organisation

With a total of 1100 employees globally and approximately 160 R&D resources – about 50 of them are embedded software developers - on 4 development sites located in four different countries Danfoss Drives has a strong technical lead in the frequency converter market.

Product development is addressed by a matrix organization where projects are carried out through integrated product development with dedicated personnel and market targets. Line organizations provide skilled developers for projects to draw upon to ensure application and domain knowledge for timely product releases. Since 1968, when a frequency converter was a pure electrical and mechanical solution, the software parts of the organization and product variation have increased significantly. Almost any functionality in a frequency converter is now handled by corresponding software.

Product releases are typically every quarter or half yearly for different product lines. The product release cycle may vary by customer request as the demands for customer-adapted frequency converters are becoming more prevalent than general application solutions.

2. A Bit of History

Interest in improving the efficiency of software development is not new. Ideas of software reuse, more efficient development and handling of product variants have been around for more than 10 years within Danfoss Drives. Inspiration has come from four areas: object orientation, software architecture, software reuse and software product line development.

The first step was to switch to object-oriented development using C++. The creation of a framework and the release of products based on this architecture happened in the late 1990s. This architecture defined a flexible architecture used now in many frequency converter products from Danfoss Drives [1].

Danfoss Drives was aware that the introduction of object-orientation and software architecture alone would not give the desired reuse. So conference visits and literature study about software reuse and later software product line development was undertaken. The idea of managed reuse was also heavily promoted within the development organization. The result was a technology project that analyzed the feasibility of managed reuse for all areas of the product development of the company. Among others, the experiences of the consumer electronics division of Philips was used in this work, most importantly regarding hardware development [2].

So when the development of the new product series began around year 2000 it was a company strategy that a much higher level of reuse should be achieved.

This strategy was applied most effectively to the development of the frequency converter hardware (PCB and mechanical parts). A rigidly followed platform strategy is used to control these parts of the system in order to ensure maximum possible reuse and smallest possible change to mechanical and electrical designs.

However, such strict reuse was not enforced on the software architecture and the code base. While all products basically reused the architectural framework, since it provide the necessary amount of freedom and variability, the evolution of the code base was not really controlled. As before reuse between projects was done in the “clone and own” way. Code was taken from project branch to other project branch via merging in the version control system.

The organization was not happy with this. But since all resources were needed for development of the first product in the new series, it was decided to live with this situation for a while.

2.1. The Breakpoint

In 2005 management and developers were ready to take the next step towards managed reuse. The first product in the new series had been successfully released. The second product cloned from an early release of the first was close to release. So resources could be found.

The company also faced the need to develop an increasing number of specialized products, e.g. frequency converters with a tailor-made user interface and also control algorithms for specific market segments.

The Embedded Software Platform (ESP) project was formed as reaction to this challenge in 2005.

3. Get Going

3.1. Organizational Moves

The migration was started as an internal project called the Embedded Software Platform project. Its aim was to find a way to create a common software platform for all products based on the object-oriented framework. The importance of this project was visible from the high profile of the initial team members. All had several years of development experience within Danfoss Drives. The team consisted of six persons. They were architects, function developers and test experts. Some of them had a large portion of their time allocated to this project; others only participated in meetings. All in all this amounted to about 3-4 full-time persons. In order to learn from others, it was decided to work closely with an external product line expert. His role was to coach the team and monitor the progress and the decisions made by the team and to help prevent common mistakes.

The project was intended to take the first steps within a relatively short amount of time (about 10 months). If the project was a success, it was intended to make it a long running activity, a so-called line organization.

3.2. Finding the Right Way

However the expectations of the project were high but diffuse. It seemed that the initial (naïve) idea for implementing the product line was the use of a library of components from which each product project could pick the components it wanted and use them. This seemed to be a very nice approach, since in theory it should allow the company to be very flexible with new products. But on the other hand there were doubts within the ESP project team whether this idea was realistically applicable for Danfoss Drives.

There was within the team also no agreement on the road to take for many other relevant questions. As an example of a discussion it can be mentioned that one topic was whether managed reuse was possible on the just introduced product series or whether it was something that should be introduced with the next product series.

It was therefore decided to participate in SPLC 2005 to learn about best practice for product line development, but also as a team building activity.

Participation in the conference was very worthwhile.

The most important point was that an incremental SPLE adoption could be a possible solution [2]. This was very pleasant news, since the part of team that had knowledge about software product line development before the team was formed, had the impression that product line development necessitated starting from scratch with domain modelling.

The team also understood that it had to consider not only the technical issues but also the organizational setup with great care.

3.3. Going for 100% platform

The visit to SPLC enabled the ESP project team to make decisions on the direction to go and to start the productive phase of the work.

The decision was to analyze whether building all products from a common code base controlled by a feature model was a viable solution. This approach was very much inspired by the experiences of Engenio [4].

Two tasks were clearly identified. One was to derive the common source base. The other was to build the feature model.

To derive a prototype for the common source code base, the code branches of the first two product projects (which were also two of the major contributors of functions) were put together. The goal was to identify common parts and also variation points from this merge. In order to get the code together quickly and to provide both projects a working instance of the platform, the merge was done in a way that all relevant differences were selectable via compiler switches. The GNU diff program is able to generate such a merge with a simple command line. It is still necessary to have a look at all merged files but in most cases the switches were introduced correctly. All in all 281 files were unified and they had 3165 #ifdef PRODUCT_X chunks. 1003 files were identical in both products, 7 files were only present in the first product and 179 were only present in the second product. Finally 19 files that existed in both products were so different, and by nature so, that it made no sense to unify them.

For building the feature model two approaches were tried in parallel. They could be called a top-down approach and a bottom-up approach.

The top-down approach was to make a feature model for two products as a kind of domain analysis. Requirement specifications, product manuals, marketing literature and lists of the configuration items (parameters) available for the user were used as the main inputs for the model.

The bottom-up approach was to build the feature model from the variations found in the existing code base. The above described unified code base was used as the basis for feature-oriented refactoring. Differences were analysed and the initial switch condition (PRODUCT_A / PRODUCT_B) were successively replaced with switches that related to product features instead of product projects.

It turned out, that the bottom-up approach was much more effective than the top-down approach for building the feature model.

The problem with the top-down approach was that although the feature models made sense, it was hard to tell whether the feature would serve well for inserting variation point in the source code. The reason for this was mainly that the feature definitions tended to be too fuzzy and it was hard to decide on the correct granularity of the features.

3.4. The Big Split

However, after having worked with building a feature model and refactoring the prototype for a while (two months or less) the idea was born, to make a fast release of a partial platform based on the prototype as a first step, while still having the long term goal of a 100% platform. There were several arguments for this.

Refactoring the complete prototype would take a long time to finish and the team faced the problem that while the refactoring took place the projects kept working on the (original) files in order to stay in line with their development plans. So when the refactoring of the prototype was completed it would become a big task for the ESP team to update this with all the changes from the two projects.

A partial platform was also seen as an advantage for the product projects. Parallel and independent finding and fixing of the same bugs could be omitted. Furthermore refactoring had found places where the two products had grown apart where they were not allowed to, since they would no longer have the option of using the same set of add-on modules if this happened.

Finally the ESP team was also very eager to get its results used by the product projects and get feedback from these.

The selection of the part of the system to be put under platform control was rather simple. Based on the existing knowledge and discussions within the ESP team a coarse split between the initial platform-managed and the still project-owned system parts was made. The division was done on a subsystem basis and the selected subsystems were those where the team felt it was important not to let them grow apart and have unmanaged variations. It turned out that about 60% of code files should become platform-managed right from the beginning. The idea was to increase the platform-managed parts later to much more than 60% but not to worry about it for now.

3.5. Changes to the development process

With the content of the minimal platform and the variation mechanisms for using it in place, it was time for the team to think about the maintenance and evolution of the platform, i.e. the way the platform once brought into the project teams should be further developed. This is a major question since a product line platform has to constantly be extended in order to support new features demanded by the customers and also since no software is entirely free from defects.

After looking at different scenarios, it was decided to keep as much of the existing development process as possible and only to make changes where it was really needed.

It was therefore stated at a high-level; that projects are still responsible for the quality of their product and will be able to release when needed. They are also responsible for fixing bugs found in their product. A project producing a feature also has to make the tests available together with the new or changed features to projects that want to a reuse the feature.

Other high-level guidance was that projects must ensure that changes made when adding new features do not affect other products not using these features. Finally, it was stated that the ESP team is a service organization for projects, with the tasks of coordinating between projects, ensuring consistent software architecture, integrating new features into the platform and releasing new platform versions.

A more detailed approach was derived from these higher-level statements.

A feature-oriented development approach was chosen since it fitted the existing practice of separate development of features in the project plus regular feature integration. It should still be the case that features are to be developed by the project teams, even those features that are located within the platform subsystems. The idea was that this would also increase the trust of the project teams in platform code since it was partially developed by them at their own pace - at least for features where no immediate demand from other projects exists.

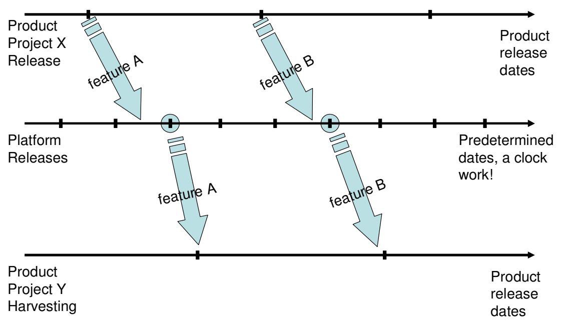

Figure 3 Feature production and harvesting

Technically each feature is developed in its own feature branch, which is merged back into the platform for integration. The additional benefit of this approach is that a project always has the freedom to use its own features currently under development if necessary, for instance, if a customer demands a quick “feature integration”. However, since the ESP coordinates feature development, its responsibility is to make sure that a feature eventually becomes part of the official platform. This is necessary in order to prevent direct exchange of features between projects (as was common before the platform). This process is depicted in **Fehler! Verweisquelle konnte nicht gefunden werden.**

Bugs should be fixed in a similar way. Once a bug in platform owned code is detected, it will be fixed in a bug branch.

Once a change to the platform (either coming from feature or bug branches) is qualified by the ESP team as fit for integration into the platform, it becomes part of the platform. But qualification criteria are not easy to define and check. One of the qualification criteria is, for instance, that a new optional feature must not introduce changes for any product not using this new feature. The question is how to check this property easily for all projects based on the platform. It was decided that in the beginning (with only two projects based on the platform) this could be handled by inspections from the ESP team, but a more resource efficient solution should be identified in the future.

4. Ready to Roll

Now the technical base (the partial platform) and the changes to the development process were in place. Based on this it was decided by the management of the development department of Danfoss Drives to roll out the partial platform at the beginning of April 2006. Roll out to the two original projects (one being developed at the main development site and one in another site) and a third project (also located at the main site) started in the preceding months was planned to happen that month. Roll out to a fourth project (located at yet another different site) was planned to happen later.

An important reason for deciding on a fast roll out was that the ESP team hit a window of opportunity. To be able to change the development process for a project, it has to be in an early phase of a release, otherwise changes would be very risky. Thus as part of the initial planning the project plans were checked and it turned out that April 2006 was a good time, since most of the projects would be in the right development phase by then. Another thing was, that with a later roll out it would no longer be sufficient to unify sources from two projects; and unifying from three or more sources would be much, much harder. This short time frame did in the end also dictate that a 100% platform was not achievable.

Two days after this decision a pre-release of the platform occurred. This pre-release was followed by a quality assurance workshop, where developers from the three projects that were going to use the platform first were reviewing and in a few cases modifying the refactoring done by the ESP team. This workshop probably partly explains why the three projects have felt confidence in the platform since the roll out.

A few working days after the quality workshop the first version of the partial platform was finally released. In the days after this roll out workshops were carried out at both involved sites. These were one-day workshops in which the ESP team trained the project members where and how the platform-based development should be done. To make sure that the workshop was effective, the timing was such that they basically ended with the setup of the project development environment for platform development. Since no new tools were introduced this was easy. The switch over was just to change the version control configuration to select the partial platform and make a few modifications to the remaining product-specific (40%) files. The ESP team had prepared these modifications in advance, so they just had to be merged in. All this went very well, the projects were able to use the platform and contribute to it.

The workshops also served other purposes beside the rather technical aspects of definition of the process, use of tools etc.. An important effect was that the project teams got in personal contact with many members of the ESP team in order to lower the barrier for communication between the projects and the ESP team. Another very important issue was to make the role of the ESP team as a service provider to the projects clear.

The effort that the projects had to use for switching over was one man-day per developer for the roll out workshop plus one to three man-days for participation in the quality assurance workshop. Of course some time has also been needed for getting used to working with the new variation mechanisms, but it is not possible to give numbers for this. The effort is anticipated to be lower than the savings coming from using a platform.

The rollout was also the start of what was called “continuous learning”. It was and is still impossible to define in advance the ultimate approach for platform development. The learning and adaptation of platform development is an explicit part of the whole operation. The ESP team felt that it was important to express the importance of this continuous learning to all stakeholders in such a development setup. This makes it easier for project teams to accept existing problems in the platform development process since improvements can be expected once problems become known.

One important building block of this learning process are so-called “Feedback workshops”, organized every few months by the ESP team, where project members and the ESP team discuss general problems with the current platform development process and also present ideas on how things could be done in the future. These meetings are different from normal coordination and planning meetings where daily work is being discussed. These have proved to be a valuable source of information for the ESP team.

Rollout to one of the development sites was delayed so that it first took place in October - 6 months later. The reason was mainly that the project located there was not able to switch over to platform-based development before that.

The workshop there also went very well, and as with the other projects the switch over was done immediately afterwards. In this case a few bugs were found, and although ESP team members corrected them very quickly, this raised a flag of concern in this project team about a lack of testing performed on the platform releases.

So, after the ESP team members returned, this project went back to a clone-and-own approach for some months. It is not entirely clear why this happened. It is true, that there was insufficient testing, but despite that, the other projects had confidence in the platform releases. The authors tend to think that the real reason is that the ESP team had failures in contacting and involving this team as much as the other project teams in the discussions about the platform. As an example, no one from this team was invited to participate in the quality assurance workshop. As a result the team members did not see the introduction of platform development as a common project for the organisation as much as others did and so lacked for good reason the trust necessary to switch over to the platform at this time.

5. Learning to walk

About two months after the mostly successful rollout – in December 06 - a feedback workshop was held at which all projects were present as well as some higher management. Shortly after this workshop the external consultant conducted several person-to-person interviews with different platform stakeholders as part of an ESP operations evaluation.

The results were all-in-all very positive, the majority of the organization believes, even at this early point, that the platform migration was already showing benefits like improved code quality and better coordination of development effort between the projects. Everyone expects to see even more benefits in the near future especially regarding development effort. However, on the other hand all of them saw room for improvement in ESP operations. Basically there were two key points: Increasing the platform managed code base and improved communication and documentation from the ESP team (process descriptions, feature plans, role descriptions …). Some of the findings in this phase will be described below.

It turned out that the initial rollout was one thing, keeping a platform alive is another matter. An important factor for this is that the platform development, which by nature requires more coordination between stakeholders, responds quickly enough to change requests raised by the project teams. This is a crucial point. If the platform changes too slowly, projects have to find ways to circumvent the platform in order to deliver what the customers want. If this situation arises on a large-scale the platform becomes obsolete. Since it was not really known how fast this integration of changes should be, in the beginning there were different time periods between the releases and they were basically made on demand. The time between two releases was 1 month to 3 months (during the summer). Another unknown factor was the effort required by the ESP team for making a release.

After the first 4 releases, the release cycle was discussed with the projects. The ESP team suggested a strict monthly release cycle in order to allow synchronization of projects to platform development. However some projects argued for a much longer release cycle (3 month to 6 months). These were the projects that expected to contribute a lot of features to the platform. Since they develop these features on their own and have access to them before the platform integrates them, they wanted fewer disturbances by new platform releases. Projects that expected to benefit from the platform were interested in shorter release cycles. In the end a monthly cycle was agreed upon. The monthly platform release is done on the third Wednesday each month, a release candidate is published one week before, and the Friday before this is the deadline for projects to submit their input. The basic timing for the releases in a year is shown in Figure 4

wk	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	▧			▧	▧			▧		▧		
2	▨	▧	▧	▨	▨	▧	▧	▨	▧	▨	▧	▧
3	▩	▨	▨	▩	▩	▨	▨	▩	▨	▩	▨	▨
4		▩	▩			▩	▩		▩		▩	▩

▧	Product project feature delivery
▨	ESP Release Candidate
▩	ESP Release

Figure 4 Platform Release Cycle

Another concern was the visibility and communication of the ESP team with the project teams. As mentioned before, in the feedback workshops and internal ESP team reflection workshops it turned out that the roles and responsibilities of the ESP team were not clear. One reason for this lack of communication was the structure of the team. It consisted primarily of highly experienced developers who were focused towards inbound communication (bug reports, feature requests, planning information, etc.). However, in the context of platform development the ESP team had to also produce outbound communication (asking for feedback, actively getting and distributing information about project plans and about the platform etc.). Since this problem was detected early on, the ESP team itself started to learn to be more outward facing. Part of this was an explicit definition of roles and responsibilities of the team and its individual members and making this known to the Danfoss Drives development organization. Another part was the formal definition of a communication plan, i.e. a list of what kind of information will be released regularly and how interaction with the ESP team within the Danfoss Drives Software Development will happen.

6. Tools

Tools and their use play an important role in product line development since, depending on the amount of variability and the number of products derived from the product line, specialized tools for handling this aspect might be necessary. However, each change in tools causes not only additional cost for the tool itself but also for training and integration into the workflow. Since starting platform development itself is a risky business, it can be a good idea to differentiate between immediately-required tools and tools which must be brought into the game at a later stage. The ESP team discussed this issue in the early stages and decided that the existing tool set should be sufficient in the beginning.

6.1. Now …

Based on a wish to minimize changes for projects it was a goal to try to use the existing and already known tools also in the platform development. The relevant development tool chain consists of a relatively straightforward embedded systems build environment based on gcc and make. Configuration management is handled by ClearCase [5], defect and change tracking is done with ClearQuest [6].

Configuration of the system is partially done by #define in header files and by inclusion of files into the build via makefile includes.

Until now the necessary project-specific configuration of the build configuration was created and partly maintained by the ESP team as a service for the projects. This was feasible since the configurations for a specific project did not change often and the number of different configurations to maintain was relatively low (about half a dozen)

6.2. … and Then

After the rollout, with the increasing number of features being available in the platform, and with many new features under development, the situation started to change. Specialized tools for handling the variability now come into focus in order to be able to generate the projects' configurations automatically based on feature selections. Another related aspect is the planning and monitoring of feature development activities in an easy manner for all stakeholders.

To achieve this Danfoss Drives is considering pure::variants [6] in combination with a project planning tool as additionally required tools to be able to scale up platform development to more projects.

A very important tool-related element is support for fast and efficient automated testing of platform-based products. This is necessary for instance in order to evaluate quickly the effects of critical changes in the platform. The existing test environment is tailored to testing activities related to longer product development cycles with explicit testing phases. However due to the short release cycles for the platform, continuous testing in parallel with development is necessary. Here an environment based on LabView [8] and TestStand [9] is currently being created.

7. Conclusions

At the current time (just after 8 months) the direct financial benefits were not yet directly visible in the balance sheet of Danfoss Drives. But on the other hand no extra effort had been spent on development, the allocated manpower was about the same as before the start of platform development. All projects were able to deliver normally according to project plans. This means that within 8 months the initial additional effort for creating the platform; the maintenance and coordination effort, had been offset by the reduced effort for the projects. This will change in the near future, with the rollout of new products, since it is expected to spend much less effort for those (derived) products than would have been necessary without platform development. Part of the acknowledgement of the achieved result from the management side was a hoped-for change of the organizational structure. The ESP project became a line organization that provides services to projects for a possibly infinite duration (as opposed to projects, which always have to have a specified end).

Analyzing the key factors for success, it boils down to: massive support of the chosen path throughout the organization due to existing positive experiences with platforms in other domains within Danfoss Drives; the existence of a suitable, well accepted software architecture and the construction of the ESP team from highly-motivated and experienced developers.

It might be worth mentioning, that the stepwise migration where each single step was small enough to be easy to understand and with points of return in case of serious problems was part of the underlying approach and proved to be suitable. The environment and the processes should be kept as simple as possible.

The integration of all stakeholders in the learning process and the open communication between them is also very important, since nothing is perfect. But if shortcomings are understood by all, it is much easier to find and live with a compromise.

Within Danfoss Drives the next challenges are improved communication among the stakeholders and change of processes and tooling with respect to the increasing number of variations to be managed (release effort reduction, automated quality assurance to anticipated and evaluate change effects). Due to the positive experiences up to now the ESP team believes that these challenges can be mastered if the same care is applied as in the migration phase.

8. References

[1] H.P. Jepsen, and F. Nielsen, “A Two-Part Architectural Model as Basis for Frequency Converter Product Families”, *LNCS 1951: Proceedings of the International Workshop on Software Architectures for Product Families*, Springer Verlag, London, UK, 2000, pp. 30-38.

[2] Internal workshop at Danfoss Drives with Dr. Jacques van Nieuwland, former General Manager and Chief Technology Officer of Philips Consumer Electronics' Audio Business Group. December 1999

[3] J. H. Obbink, K. Pohl (Eds.): “Software Product Lines, 9th International Conference, SPLC 2005”, Rennes, France, September 26-29, 2005, *Proceedings. Lecture Notes in Computer Science 3714* Springer 2005, ISBN 3-540-28936-4, see also http://www.sse.uni-due.de/splc2005/

[4] W.A. Hetrick, C.W. Krueger, and J. G. Moore, ”Incremental Return on Incremental Investment: Engenio's Transition to Software Product Line Practice”, *in LNCS 3714: Proceedings of the 9th International Software Product Line Conference 2005*, Springer Verlag, 2005.

[5] ClearCase Homepage: http://www-306.ibm.com/software/awdtools/clearcase/

[6] ClearQuest Homepage: http://www-306.ibm.com/software/awdtools/clearquest/index.html

[7] pure::variants Homepage: http://www.pure-systems.com/pv

[8] LabView Homepage: http://www.ni.com/labview/

[9] TestStand Homepage: http://www.ni.com/teststand/