

Life Cycle Cohesion: Roadmap-Based Software Architecting for Optimal Software Evolution

Jacco Wesselius
Jan Willem van den Beukel
Wim Pasman
Joland Rutgers

Philips Medical Systems
December, 2007

Abstract

For products with a long development and economic life cycle, software evolution is a powerful approach. Software evolution requires a sustained development and release rhythm. This rhythm is often broken due to large-scale re-architecting activities needed to maintain the quality of the software or to implement new features. To avoid being hit by a rhythm-breaker, an organization's mid/long-term view on the future (its roadmaps) can provide a valuable source of information. Based on this information, the software architecture can be optimized. One of the attributes to optimize is: reduce the level of coupling between items with different evolution rhythms (maximize "life cycle cohesion") and isolate areas where revolutionary developments are expected. In this paper, an overview is given of the roadmapping approach in Philips Medical Systems and on the way its deliverables influence software architecting.

1. Introduction

In [1], we discussed the dynamics of the innovation roadmap of the software in complex medical equipment. We used this case to make an inventory of the technology eco-system [2] (see figure 1) in which the software evolves. We concluded that the evolution of the software in the equipment is influenced by many factors which are not related to the software or to software technology. Software evolution is frequently disrupted by innovations in other areas like: X-ray image detectors, system architectural changes, and hardware innovations that create completely new requirements and that offer new technical solutions. In these cases, often a "software revolution" is asked for.

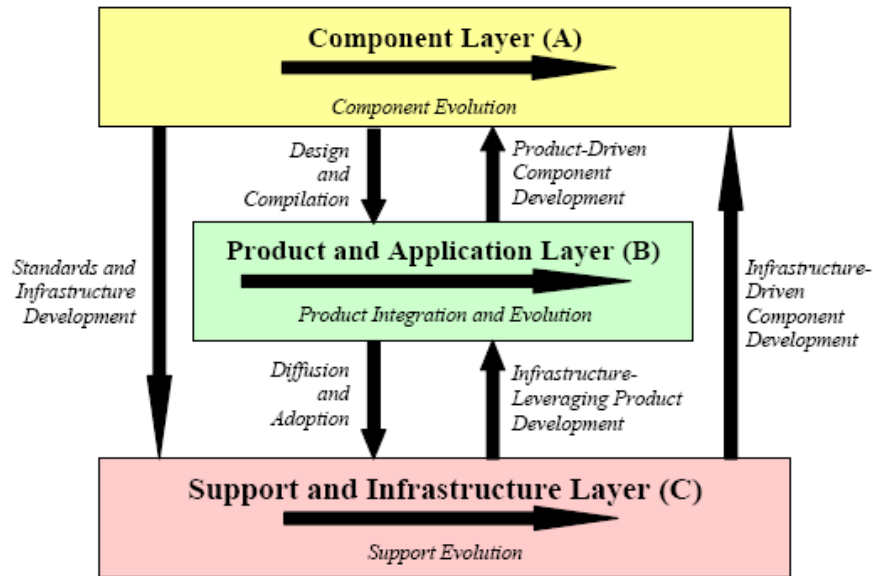


Figure 1: The Technology Eco-System (from [2])

We concluded that in order to optimize the software evolution in the equipment we studied, a roadmapping process is needed to guide the software evolution. This process would have to provide early warnings for changes that might result in software revolutions. When these potential revolutions are known well in advance, an evolutionary approach could be used to prepare for the introduction of new technologies. This way, the revolution could be avoided or at least be limited in scope.

In this paper we will look into our roadmapping processes. First we will look at evolution from a product line and software platform development perspective, since product line development complicates software evolution by creating many dependencies among platform builders and platform users. After this, we will give an overview of the roadmapping process which is typical for the approach in several business units of Philips Medical Systems. After this introduction we will discuss how to connect the roadmapping process to the evolution of our software. Finally, we will look into life cycle coupling and life cycle cohesion as two important concepts to assure that architectural choices are in line with our policy to facilitate optimal software evolution.

2. Challenges in Evolutionary Software Platform Development

In Philips Medical Systems, we apply product line development and software platform development principles for the software in our medical equipment (see section 15 of [3]). Developing a software platform adds complications to evolutionary software development. Apart from many technical and architectural complications, planning complications are a main issue. Since software platform development is an essential element in the context of our work we will first

explicitly address some of the planning issues involved in evolutionary development of a large scale software platform.

For an investment in software platform (large-scale reuse) to be justified, the platform needs to have an adequate number of users. A well-known heuristic is that at least 3 users are needed to justify the investment in reuse (see for instance section 6.1 of [4] and chapter 14 of [5]). For the investments to be really economically justifiable, we aim for a much higher number of users of the platforms we develop. When the number of users grows, the number of commitments to be made by the platform development group (in terms of content, quality, and delivery timing) grows at least as fast.

When system groups start using a software platform, they become dependent on it. To release their systems with the features and quality they promised to the market, they need these features to be supported by the platform and they need the platform to be reliable, maintainable, etc. in line with their overall quality requirements. This means that they ask for commitments from the platform development group. While the platform is evolving, they need stable, baselined, released versions of the platform for them to be available at the right moment in time. For them, a released version of the platform is fully tested and documented. It comes with a known set of features and with a known quality.

The issue: releasing the platform this way requires serious investments and cannot be done too many times per year. Typically the software platform is released twice per year. The development projects and the release cycles of the system groups will be coupled to the release rhythm of the platform on one side, and driven by the rhythm of the market (especially large trade shows) on the other side. When these two rhythms are out of sync, the evolution rhythm of the platform will get in the squeeze. Planning the evolution of the platform and planning the corresponding release content, quality and timing is a major issue when the number of platform users grows. The success of the platform may become its stumbling-block. In that case, when the platform evolution stumbles all its users fall flat on their face.

One of the ways we chose to reduce the planning problem is to involve the users of the platform in its development. By applying open source principles inside the PMS organization (inner source software development), we try to give the platform users more grip on the development of the platform. Users can contribute to the platform's growth and quality. Users can contribute their specific knowledge and competencies to the platform development. In short users of the platform are not completely dependent on the choices made by the platform development group, but they can steer the development of the platform in a direction that fits their specific business needs.

To make sure that platform users start making contributions to the inner source software development activities of the platform, and to make sure that system

groups start using the platform we are studying market mechanisms and business models for the inner source software market (see [6]). To approach the bazaar-like development culture of the open source communities, while making sure that the overall business goals of Philips Medical Systems are achieved we have to keep a balance between the bottom-up evolution of open source software development and the top-down planning culture inside a large organization.

Market mechanisms and business models on the inner source software market are mechanisms to use. A clear roadmapping process that is well linked to the software evolution process is another one.

The roadmapping process should make sure that the longer-term goals are clear and that the overall Philips Medical Systems strategic goals are being worked on. All initiatives taken in the inner source software development processes should be reviewed with respect to this roadmap.

Furthermore, the roadmapping process should give information on the requirements the various users have for the platform. When the number of users grows, the likelihood of conflicting requirements grows. But also the likelihood of similar future requirements from several groups grows. Avoiding conflicting development activities and enhancing synergy between groups with similar requirements is one of the crucial planning activities needed to guide the evolution of the software platform.

Finally, we do have a central group which is responsible to the software architecture of the platform. This group does the core of the development work of the platform. Optimizing the added-value of the work done by this group requires a longer term view on the future of the platform as well. The inner source-based evolution of the platform should not damage the integrity of the platform architecture and it should not be in conflict with the architectural work done by the central platform group.

Based on the observations above, we conclude that the evolution of our software platform takes place in an environment where many factors influence its development. On one hand we have to deal with non-software developments in the software's technology eco-system (see [1]); on the other hand we have to deal with the evolution of software developments in the inner source environment of the software platforms we use. To enhance the value of our evolving software, we need to invest in roadmaps to guide its evolution.

3. Roadmapping Processes

In large scale development processes, the roadmapping process is one of the strategic processes. The roadmapping processes aims at integrating longer term visions from various disciplines into a consistent and aligned set of business roadmaps. The resulting roadmaps are input to the development programming process which defines and executes a set of projects to deliver the desired

assets (processes, skills, intellectual property, products and product features etc.).

By investing in roadmapping, an organization aims at making sure that activities are started in time. If, for instance, a 2-years research project needs to be done to develop technology for a product to be delivered next year, the organization should have started that 1 or 2 years ago. By creating a roadmap which plots the expected product releases several years ahead, and by combining that with an overview of the running and required research projects, the organization can decide which research projects to start and which to postpone. Similar overviews can be created for the required competencies, for the preferred way of working, etc.

To be successful, the roadmapping process should cover all aspects of the technology ecosystem [2] (see figure 1):

- Which components/basic technologies can be expected to be available within the roadmapping time horizon?
- Which supporting technologies are expected to emerge that interact with our products? Which demands are expected to arise from the emerging supporting technologies and the emerging infrastructure?

The roadmapping process is (in essence) relatively simple. An overview of the roadmapping process used in one of the X-ray business lines in Philips Medical Systems is sketched in figure 2. This process focuses on one of the aspects of roadmapping: aligning the technology options with the market demands. Similar processes can be sketched for the alignment of other aspects.

The process consists of four steps (often executed iteratively). These should be performed keeping the complete technology ecosystem (figure 1) in mind:

- **Plotting Feature Demands.**

This process focuses on understanding the market in terms of values, requirements and features needed to make the right value proposition. In order to do that, the key decision makers and other key players in the value chain are being inventoried. To understand their decision-making process, a close relation with key customers and key suppliers is maintained.

The output from this marketing-driven process is an overview of valuable products and product features. It identifies the time period in which features are valuable, and it indicates the value of those features in the key market segments.

- **Plotting Technology Offers.**

This process focuses on understanding the technology domain. During the process, a map is made of the expected technology status in the key technology areas. Questions being considered are: when will existing technologies become obsolete? which new technologies are expected to

emerge? For emerging technologies, the timeliness, the risks and the costs of developing or adopting the new technologies are being estimated.

The key question to be answered during this process: what is the potential of new technologies for creating value for our customers and for ourselves? In addition to that, the question is answered: what needs to be done if we decide to use a technology opportunity to create the expected value?

Due to the iterative nature of the process, outputs from the feature demand plotting process may trigger potential technology offers, and vice versa. The alignment and priority setting process below will play a pivoting role.

- **Alignment and Priority Setting.**

The first two activities create the inputs for the decision making process. In this process, the technology offer is mapped with the feature maps: where can technology add to our value propositions? Where does our technology offer need improvement to support our value propositions?

Furthermore, a reality check will be done. Based on a first order estimate of the investments needed to develop a product feature, the business case for building the product feature will be evaluated.

The outcome from this process is basically an overview of the value propositions we wish to pursue and the investments in R&D needed to realize these.

- **Plotting the Aligned Roadmaps.**

Finally, the outcome from the decision making process is translated into a set of aligned roadmaps: product roadmaps indicating features of future products etc.; technology roadmaps indicating the investments in new technologies.

The emerging roadmaps are used to communicate the vision to stakeholders inside and outside the organization. Furthermore, they are input to managing the product and technology portfolio of the internal product groups. These product groups execute the projects needed to realize the roadmaps.

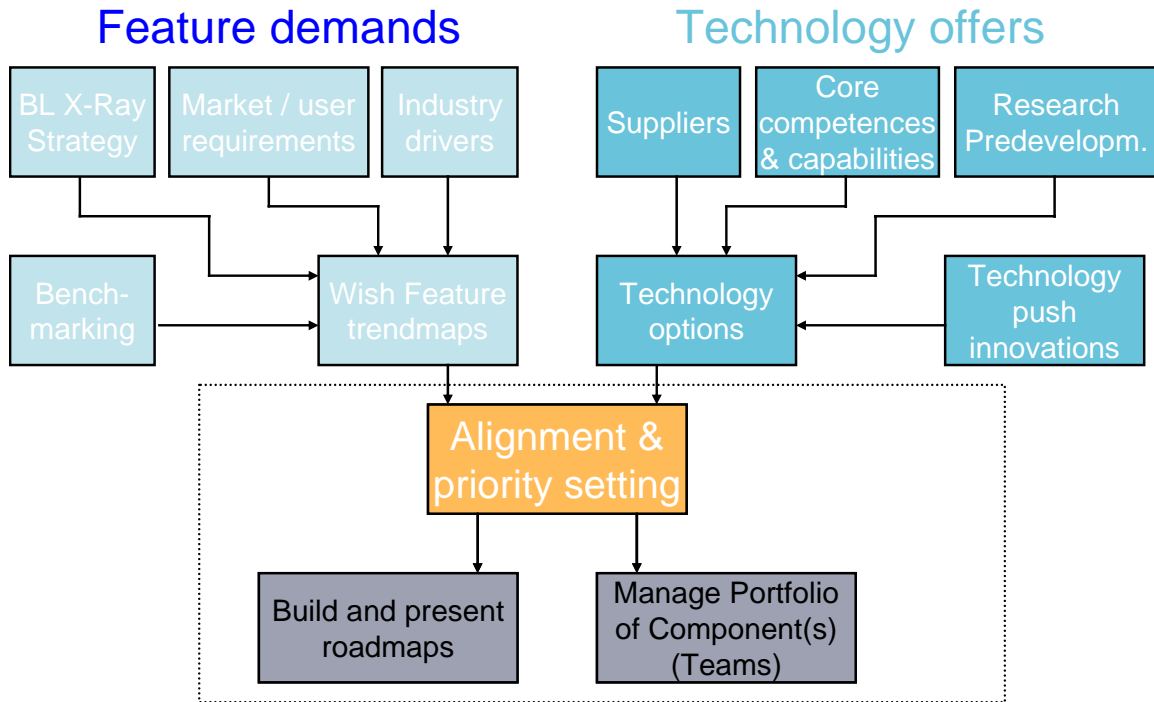


Figure 2: Technology Roadmapping Process Example

4. Linking the Roadmap to Software Evolution

For the software in our equipment, the outcome of the roadmapping process as sketched in the previous section will be a set of changes and extension that have to be made to the software in the short-term and mid-term future. To optimize the evolutionary development of our software (i.e., to avoid large-scale redesigns which would break the evolution rhythm) we will use the output from the roadmapping process in several ways. In this section, we will present several ways to use the output from the roadmapping process during evolutionary software development.

The central element in evolutionary development is keeping the changes small in order to keep up the “development rhythm” (e.g., chapter 4 of [7]). To optimize evolutionary software development, the central role of the roadmapping process is linked to the development rhythm.

Based on the roadmap info we look for those elements that would break the rhythm. We will address three main elements:

1. Spectral Analysis: Analyzing the Rhythms
2. Keeping the overall Rhythm: Isolating Revolutions
3. Avoid Sudden Changes: Pre-studies to Explore “the unknown”

We will discuss these briefly one-by-one. An overview of the process is sketched in figure 5.

Spectral Analysis: Analyzing the Rhythms

One way to complicate evolutionary development is life cycle coupling. When units with different life cycle rhythms are tightly coupled, it will be very difficult to implement evolutionary software development, since these subsystems cannot evolve with their own rhythm. Large scale developments will result. To avoid this, the software needs to be analyzed with life cycle rhythms in mind: what is the natural life cycle rhythm of subsystems and have subsystems with incompatible life cycles been decoupled sufficiently? In section 5, we will discuss architectural approaches to life cycle decoupling.

The roadmap is an essential element to analyzing the software with life cycle rhythms in mind. It gives an overview of the changes expected to be needed in the near future. When analyzing the impact of these changes on the underlying software architecture, patterns of evolution rhythms emerge: some subsystems will require frequent change; some other subsystems will be relatively stable, etc. To optimize evolutionary software development, units with: (i) a high level of (functional) cohesion and (ii) a similar life cycle rhythm should be grouped. In addition to the commonly used classifications for software cohesion where functional cohesion is the highest level of cohesion [8], we want to add an additional type of cohesion: *life cycle cohesion*.

In the X-Ray business unit of Philips Medical Systems, these observations have resulted in re-architecting our product line architecture.

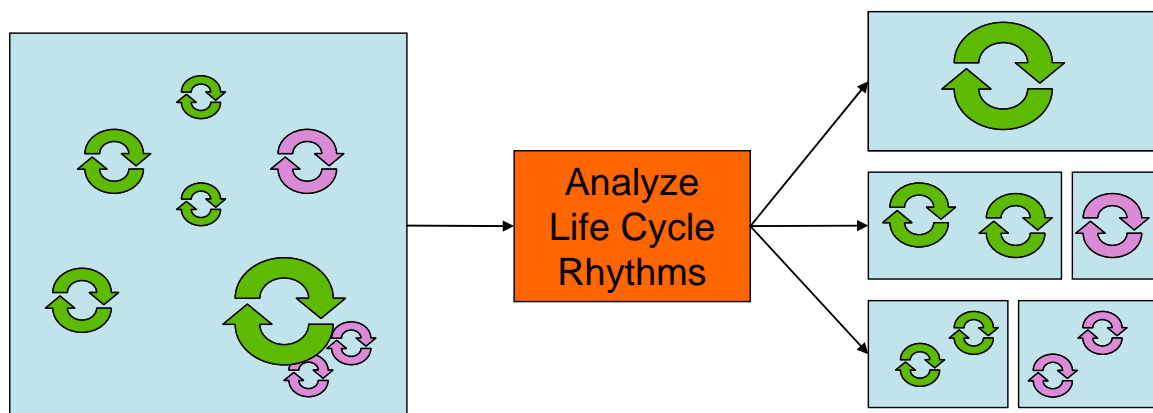


Figure 3: Decomposing software based on life cycle rhythms

Keeping the overall Rhythm: Isolating Revolutions

Innovative features that require large-scale redesigns of the software are a major threat for evolutionary software development. Making these redesigns one large project activity would break the evolution rhythm. While changing the architecture to prepare for the feature, the evolution beat would stop. This can have dramatic consequences; especially when the need for the feature becomes eminent late (see also chapter 4 of [7] for a discussion of the “Late Killer Feature Anti-Pattern”).

As many have experienced, disasters like this cannot be completely avoided, but much can be improved when the roadmap is of adequate quality and if it has an adequate time-horizon. To avoid large-scale redesigns that break the evolution rhythm, the following can be done (see figure 4):

1. The innovation content of the roadmap can be analyzed to identify those innovations that are likely to result in large-scale redesigns
2. To prevent these large-scale redesigns from breaking the evolution rhythms, some small-scale redesigns can be planned to “isolate the revolution”. The area that is expected to undergo a major redesign is encapsulated with well-defined interfaces in a series of small steps that can be implemented in the evolution rhythm of the (sub)system they are contained in.
3. When the encapsulation has been completed to a satisfactory level, a parallel track can be started to do the large-scale redesign. While the redesign is done with its own rhythm, the rest of the software evolves with its own rhythm.
4. When the large-scale redesign is completed, the resulting software joins the evolution of the rest of the software again.

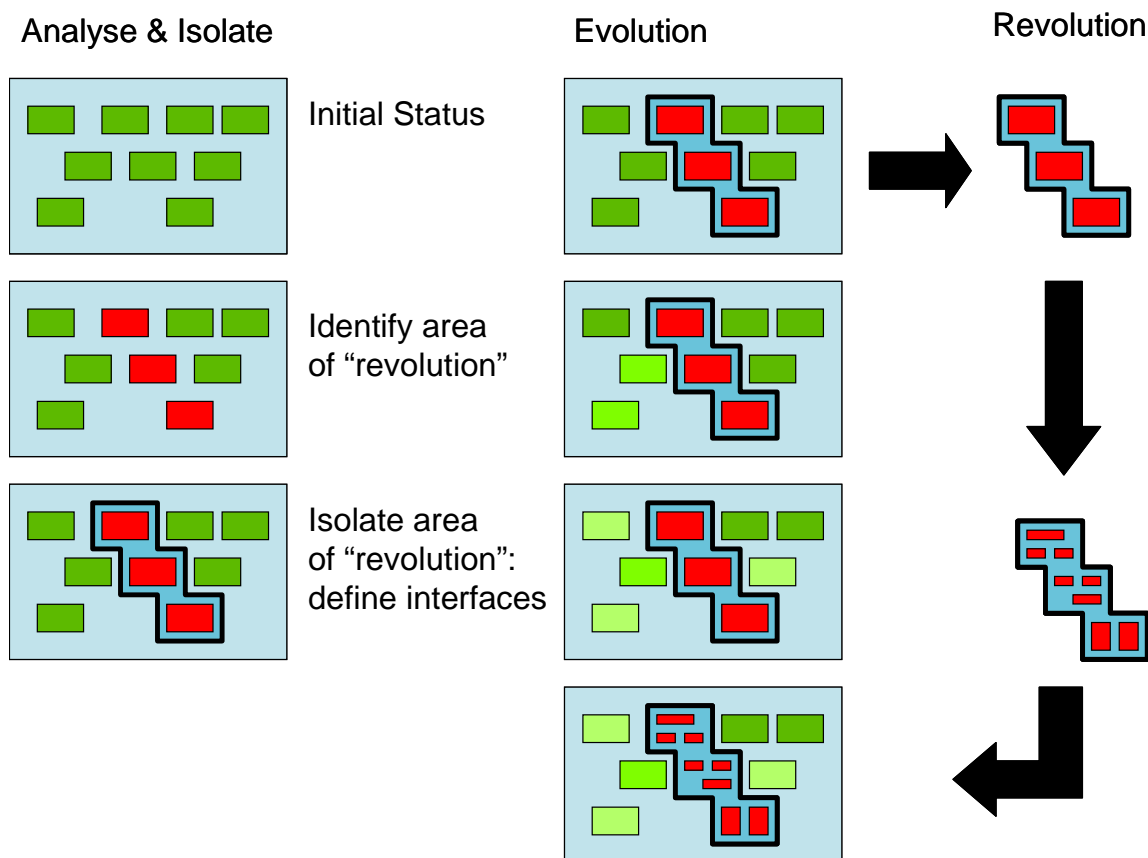


Figure 4: Isolating Revolution; keeping the Evolution Rhythm alive during a Revolution

The principle behind this approach is: keep instable, less controlled changes separate from the mainstream. The same approach can and should be applied for software components that are not fully controlled by the software development group.

To ensure that the encapsulated software can be merged back into the product after it has been modified, interface management (well-defined interfaces which are under change control with a well defined change control process) is crucial. This might introduce some overhead and it may seem to slow down the software evolution, but in order to be able to combine the results of the “revolution” back into the evolving software, it is an absolute necessity. The approach sketched above reinforces that the essence of software architecture lies in defining the interfaces among the software building blocks.

It is a well-known phenomenon that software quality tends to decrease over time (see for instance the 8 Laws of Software Evolution [9]) unless effort is spent on continuous maintenance of the software architecture. This means that regular re-architecting activities need to be done to keep the software quality adequate. These re-architecting activities often have the same type of effect as the large-scale innovations mentioned above. We think that the same approach can be followed for this as sketched above: if the maintenance activities are planned well in advance (based on a software roadmap), the encapsulation approach can be considered for this just as well.

The value of using the roadmap in the encapsulation process is that it gives a longer term view on potential areas of instability. Knowing these areas in advance, allows us to do the encapsulation in an evolutionary manner in order to be prepared for “the disaster” before it happens. If this is applied successfully, large-scale redesigns do not become rhythm breakers.

Avoid “Sudden Changes”: Pre-Studies to Explore the “Unknown”

In many cases, the consequences of roadmap items on the software architecture are not completely known in advance. We have to deal with this uncertainty as early as possible. Since we do not know the impact of the roadmap item, we cannot prepare the architecture in an evolutionary fashion as discussed above. This means that the roadmap items that are not well understood increase the likelihood of colliding with a rhythm breaker. In those cases, we therefore start pre-studies as soon as possible to increase our understanding of these roadmap items and to study their impact on the software.

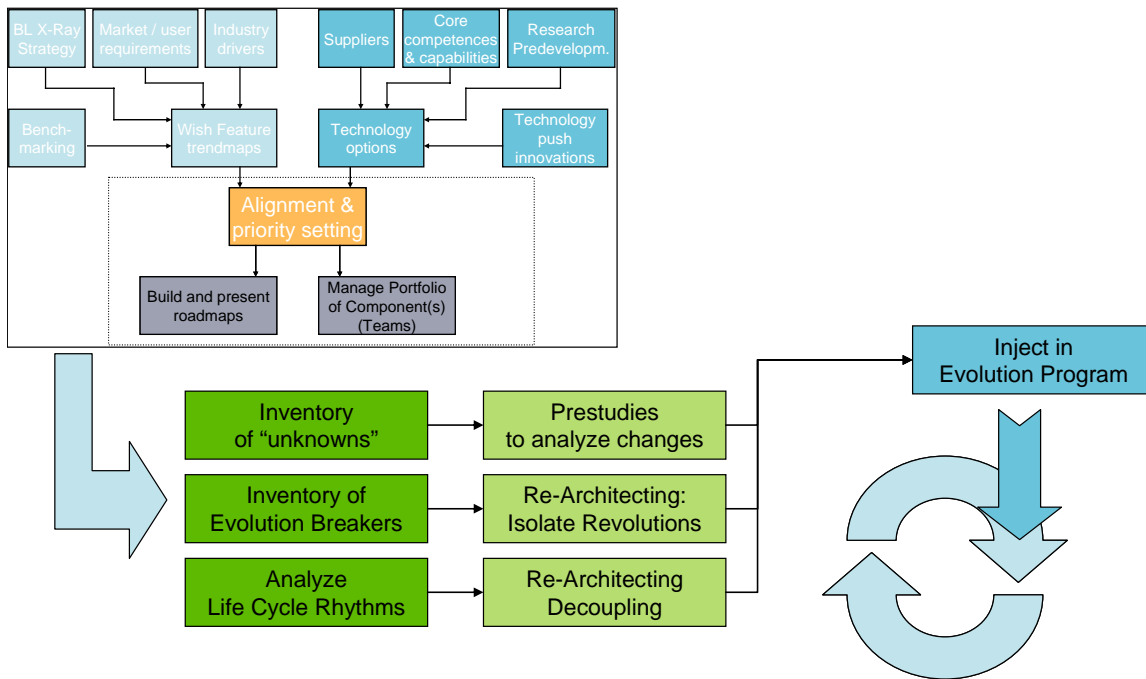


Figure 5: Linking Roadmapping to Software Evolution

Summarizing, the link between the roadmapping process and evolutionary software development is as follows:

1. based on the roadmap, potential rhythm breakers are identified;
2. for these rhythm breakers, architectural changes will be proposed to: (i) decouple conflicting rhythms, and (ii) to isolate revolutionary changes;
3. the architectural changes are split into a series of smaller development steps that can be injected into the evolutionary development process without breaking the rhythm.

The resulting evolution plan could look like figure 6. You see that Pre-studies and platform evolution steps are combined in a plan with evolutionary release steps.

Essential to this approach is that enough time is left between observing the potential future rhythm breakers and being hit by them. This time is needed to implement the architectural changes in a series of smaller steps. Therefore, the time scope of the roadmapping process should be set to an adequate time range. Short-term planning cannot avoid a collision with one of the rhythm breakers.

The right time-scope of the roadmapping process is organization dependent. We cannot give the right value in this paper. But, any organization seeing its evolutionary development rhythm being frequently broken should seriously consider: (i) taking the results of its roadmapping process more seriously, and (ii) extending the planning horizon of its roadmapping process.

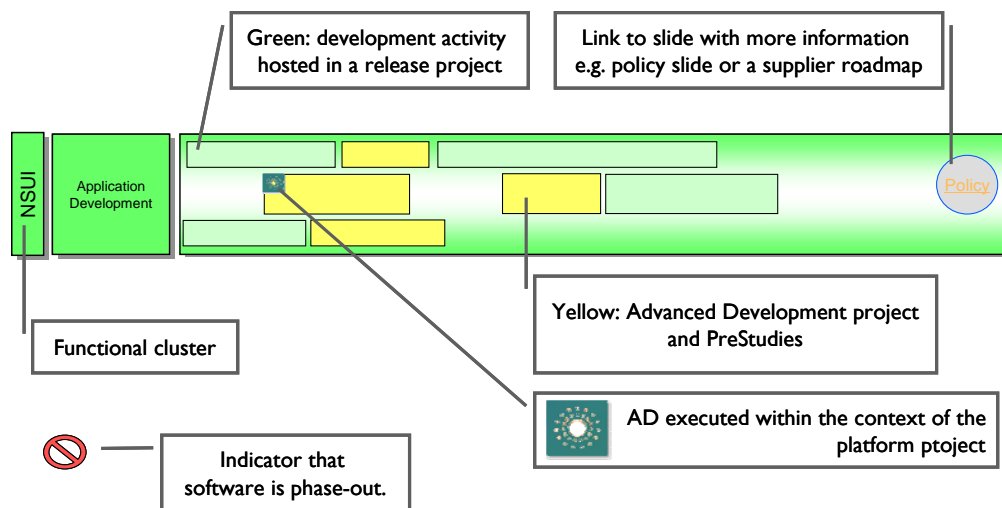


Figure 6: Example of a Software Evolution Plan based on Roadmap inputs

5. Optimizing Software Evolution: Software Architecture

A central element in our approach is: decoupling the life cycles of software subsystems that have different evolution rhythms. In our efforts to reduce the level of coupling, we have used the following definitions:

Cross-Unit Requirements Some features have a system wide impact: only if changes are made to more than one unit, the feature can be offered. These *cross-unit requirements* are inevitable. They are a potential source of life cycle coupling.

Requirements-Coupling Requirements-coupling is introduced if one subsystem can only function when other subsystems offer certain features. This is often caused by Cross-Unit Requirements: to implement the Cross-Unit Requirement, one subsystem imposes (functional) requirements on the other subsystems which are not covered by their interface specification.

Design-Coupling Design decoupling means: for one subsystem to function properly, another subsystem needs to have a certain *internal* design.

Implementation-Coupling Implementation-Coupling means that two subsystems share the underlying platform, operating system or hardware. Changes to these will result in (potentially) changes in more than one subsystem.

For each of these three types of potential coupling, the question should be asked: does the coupling introduce coupling between entities with an evolution rhythm that is different now or that will be different in the future? In that case, changing the software architecture to enhance the decoupling between these entities should be considered.

In software architecture literature, one of the attributes of software that characterizes its quality of its design is *cohesion*. In [8] an overview is given of types of cohesion. This overview runs from “coincidental cohesion” to “functional cohesion” based on the calling dependencies of a set of functions. According to this classification, the best type of cohesion is “functional cohesion”. In this case, a set of functions is grouped together that all contribute to a single, well-defined task.

In this overview and qualification of cohesion types, the aspect of evolution rhythms is not addressed at all. In view of our ambition to enhance the evolvability of a software design, we have experienced that this aspect should not be ignored. When assessing a software architecture for its evolvability, we have therefore coined the term “life cycle cohesion” to express that a group of units is grouped (and coupled) that have the same evolution rhythm. To safeguard the software evolution process, “life cycle cohesion” is the most desirable cohesion type.

Since life cycle coupling considers elements different from those considered for the cohesion types discussed in [8], we cannot say that “life cycle cohesion” is better than “functional cohesion”. For optimal software evolution, the preferred architecture would have both “functional cohesion” and “life cycle cohesion”. The task of the software architect is to balance between those two aspects and find the optimum combination of the two.

6. Conclusion

In Philips Medical Systems, we are building products that have a long life time. The software in these systems typically evolves over a period of many years. We therefore adopt an evolutionary software development process for many products. The evolution has a relatively low rhythm of external deliveries. But the internal rhythm is much higher. For both the external evolution and the internal evolution we want to optimize the evolution process of our products. This means that optimizing the software architecture for optimal evolvability is one of our concerns.

In this paper we have sketched our approach to use the long-term view reflected in roadmaps to optimize software architecture in an evolutionary way. We have expressed that “life cycle cohesion” is an important aspect to consider when assessing the quality of a software architecture. In the literature of software architecture, the well-known classification of cohesion types does not address this issue.

In PMS we are reconsidering the reference architecture of our X-ray systems to optimize life cycle cohesion. In this effort, we explicitly take the expected evolution rhythms (as expressed in our roadmaps) into account. The resulting architecture is in many aspects similar to the existing architectures, since we still take the classical architectural qualities into account. But we do see differences. We see large sub-systems emerging with different evolution rhythms. We put a strong focus on defining the interfaces between these subsystems. We expect these large sub-systems to be able to evolve relatively independently from each other. By making life cycle decoupling a design criterion, we see that design decisions get a different outcome: in “the past” it was relatively common practice to base a system design on “design coupling” between two components. We now see that design coupling between the large subsystems is not accepted any more, since this would kill the life cycle decoupling we are striving for.

We hope to be able to harvest the fruits from this redesign effort in improved time to market and reduced maintenance costs. But, as with many investments in architecture, only the future can prove us right or wrong.

References

- [1] Jacco Wesselius, Wim Pasman and Jan Willem van den Beukel, ***Software Evolution and Roadmapping: a “Technology Eco System Based Approach” to Study Software Evolution Patterns and Drivers for Software in Medical Equipment***, unpublished paper, available on ITEA Serious WIKI
- [2] Adomavicius, G., Bockstedt, J.C., Gupta, A., and Kauffman, R., 2005. ***Technology Roles in an Ecosystem Model of Technology Innovation***.
http://www.misrc.umn.edu/workingpapers/fullPapers/2005/0504_030305.pdf
[May 1st, 2007]
- [3] Frank van der Linden, Klaus Schmid, Eelco Rommes, ***Software Product Lines in Action – The Best Industrial Practice in Product Line Engineering***, Springer Verlag, 2007
- [4] Paul Clemens, Linda Northrop, ***Software Product Lines – Practices and Patterns***, SEI Series in Software Engineering, Addison-Wesley, 2002
- [5] Ivar Jacobson, Martin Griss and Patrik Jonsson, ***Software Reuse – Architecture, Process and Organization for Business Success***, Addison-Wesley, 1997
- [6] Jacco Wesselius, ***Running a Bazaar inside a Cathedral – Business Models in an Inner Source Software Market***, unpublished paper, available on ITEA COSI WIKI, accepted for publication in IEEE Software.
- [7] David M. Dikel, David Kane, James R. Wilson, ***Software Architecture: Organizational Principles and Patterns***, Prentice-Hall, December 2000 (section 4: Rhythm: Assuring Beat, Process, and Movement),
http://www.bredemeyer.com/pdf_files/PH021RythmChapter.PDF
- [8] Yourdon, E.; Constantine, L L. (1979). ***Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design***, copyright 1979 by Prentice-Hall, Yourdon Press
- [9] M. M. Lehman, ***Laws of Software Evolution Revisited***, pos. pap., EWSPT96, Oct. 1996, LNCS 1149, Springer Verlag, 1997, pp. 108-124
<http://www.doc.ic.ac.uk/~mml/feast2/papers/pdf/556.pdf>