INTERNATIONAL DESIGN CONFERENCE - DESIGN 2002
Dubrovnik, May 14 - 17, 2002.

Information Management for Multi-Technology Products

Daniel Svensson, Ivica Crnkovic

Keywords: PDM, SCM, Integration

1. Introduction

Products traditionally regarded as mechanical or electronic products, are more and more dependent on software and firmware. Software is used to implement functions that earlier have been done in mechanics or electronics, thus software is becoming an integral part of products. Examples of such multi-technology products can be found in many industries such as consumer products, telecommunications and in the automotive industry.

Companies designing and producing mechanical and electronic products often have problems integrating the mechanical and electronic design process with the software design process. The processes have become isolated islands, and the overall process faces two types of problems: First it is difficult to efficiently exchange information between different processes, and second, in many cases there is multiplication of the same information. The result is additional development or maintenance costs, delivery delays, or similar, and even the decreasing quality of the delivered products. Therefore, many companies are facing a strong need to integrate both the processes and the information systems. An overall integration of these processes is not trivial. First, the separate processes are complex themselves, and often suffer from the same problem themselves [Stevens 1998, Estublier 1998]. Second, the traditions and cultures of these areas are different. While mechanics and electronics development have a long tradition of development with many established standards and procedures, the software development field is less mature, but more flexible. When people from these areas meet, they do not understand each other. The third problem is the difficulty to integrate tools from the domains [Crnkovic *et al*. 2001]. PDM systems (Product Data Management) are used to manage information in mechanical and electronics design [CIMdata 2001], while information about the software design is separately managed in SCM systems (Software Configuration Management). The use of separate information systems makes it difficult to manage the complete product if it is built from mechanics, electronics and software. An important prerequisite for a successful process integration is the integration of these tools. For vendors and users, the payoffs are likely to be tremendous for a relatively low cost and minimal investment of resources. However, in general there is a lack of integrated knowledge of both disciplines, and exhaustive research is needed to find out which ways of integration and interaction are the most suitable.

This paper aims to address these problems. We give an overview of the mechanics and electronics (ME[1]) design process and the software (SW) process and discuss the information flow within and between them. This includes finding the common parts of the development processes and the

[1] ME (Mechanics and Electronics) design is not an established term, but used in this text.

information models; what information can be managed in separate by the teams, and what information is necessary on the product level. Based on the processes and information used, the tools and their integration is discussed. Underlying the paper is a comprehensive interview study of the situation in Swedish industry [Persson *et al.* 2001], their current use of processes and tools and their needs. The current trends and state of the art in research and industry have been reviewed, and commercial PDM and SCM tools have been studied.

The remainder of the paper is organised as follows. Section 2 gives an overview ME development process and the SW development process, and analyses the information flow in them. Section 3 gives an overview of the main functions of PDM and SCM tools. Section 4 discusses the integration of the processes, information flow and the tools. Finally, we conclude the paper in Section 5 by summarising the findings and listing issues for future work.

2. The needs

The aim of this section is to define how the processes are performed and what information is managed in ME and SW development.

2.1 The process

Product development involves many activities performed in parallel. The *Integrated product development* model [Andreasen and Hein 1987] consists of three main activities: *Marketing, product development* and *production*. Through these activities, we must be able to manage product related information. This high-level model is general and valid for ME as well as SW development, although different sub-activities and different priorities will be emphasised. For example, software production activity is significantly smaller than its development, while optimisation in the production phase are of crucial importance for mechanics and electronics production.

We shall focus on the development process. To describe the processes, we use generalised process models in which we distinguish different activities shown as a sequence. In reality, activities are performed in parallel and repeated in an unpredictable manner, but process descriptions can still be useful for several reasons; here we will use the process models to study how information is created, how it evolves and how it is used during the development process. Our approach is to describe the processes for ME and SW development and map the processes to the information types associated with product development. By studying the differences between the processes for ME and SW design and comparing the evolution patterns of the information created, we can find the necessary interfaces needed between the two processes and find out how they can be co-ordinated, planned and supported by information systems.

To describe the main steps of pure ME development, we have adopted a generic development process from Ulrich and Eppinger [1995]. The process contains five steps, as shown in Figure 1. It starts with a conceptual development phase, were the needs of the market are investigated and the overall requirements defined, followed by the generation of product concepts. In the system-level design phase, the architecture of the product is decided, which includes to identify sub-systems, components and the interfaces between them. The components are designed during the detailed design phase. The testing and refinement phase includes to build product prototypes used to test both the product and the production system. During the production ramp-up, the production system is used for serial production of the product, starting at a low pace, and then increasing to full production capacity. We will not treat the development of the production system here, but we must take it into consideration as an essential part of ME development. Figure 1 shows the types of information used to describe the product. The bars show roughly in which of the phases information evolves. In the early phases, the product is

described by functions and possibly also in terms of principle geometric layouts. The architecture is not an information object in itself, but is described by geometric layouts, system structures and preliminary product (component) structures. During the detail design, the product structures get more detailed until it is time for the final release of the product documentation.

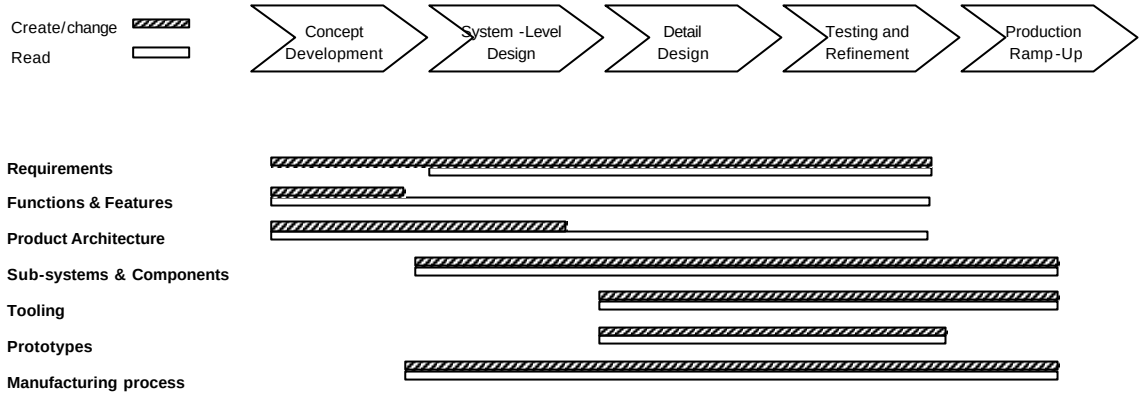

Figure 1. ME development process and information usage

The software development process [Sommerville 2001] follows a similar schema. During the requirements analysis phase, the system's services, constraints and goals are established in the system specification. In the next phase, software design, an overall system architecture is established, in which the fundamental software systems abstractions and their relationships are identified and described. The implementation phase comprises formalisation of the design by creating source code or using pre-existing code or packages. The implemented units of software are created and tested. In the integration phase, the SW units are integrated in a software system. The integration phase corresponds to the production of ME. During the test phase, the system is verified and validated. Finally, the system is released and delivered to a customer. The phases of the development process are shown in Figure 2. This sequential model is also called a waterfall model. We can, however, see that the information flow is not sequential, bur rather follow a V-shape. Indeed there is a development model called the V-model [Stevens *et al.*, 1998], which provide support for exchanging information between the different phases of the process.

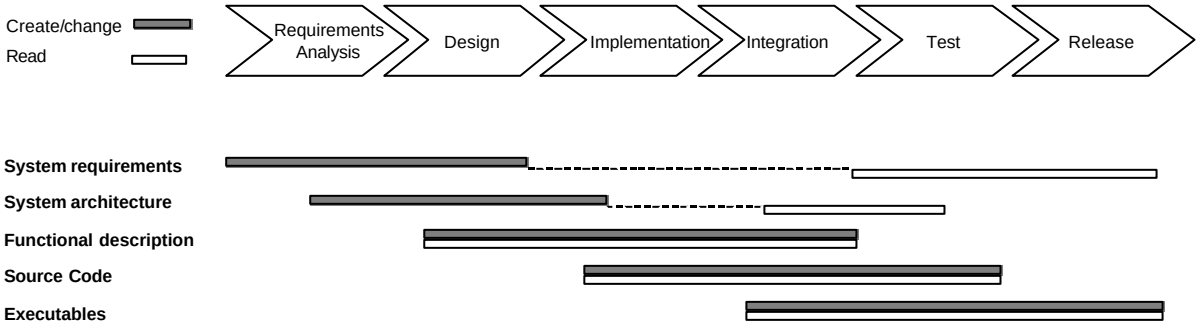

Figure 2. SW design process and information usage

2.2 The information

The information needed to capture the work during the development includes various kinds of structures and design documents such as geometry models and text documents.

The evolution diagram for the ME development process contains various information types describing the design. (We do not treat the information describing the manufacturing process here.) A requirements driven product model by Sutinen et al. contains this information [Sutinen *et al.* 2000], see Figure 3a. This information model contains requirements, function structures describing the functions and features, system structures that partly describes the system architecture, and finally part structures describing the parts (components) building the product. The model also contains property models and life-cycle models, but these are not described here. The information structures can be inter-related to describe how requirements are fulfilled by functions, how functions are solved by systems, and how systems are realised by parts. The design documents, geometry models etc., can be structured by relating them with the structures.

In the software development process, structures on similar levels of abstractions are used, but there are differences, see Figure 3b. Similar to ME development, SW development starts with requirements engineering, which includes requirements collection, analysis and specification. The result is a list of grouped requirements. The software architecture developed during the design phase deals with structure of the software system, which comprises software components, the externally visible properties of those components and the relationships among them. The development structure used during the implementation phase includes source code and documentation related to it. It is natural that the software architecture and the development structure are as similar as possible, but in practice they are separated, which is a result of two factors: First, the development process itself. The isolation and co-ordination of development activities requires a structure different from the software architecture. Second, many of the development tools used require specific structures. A completely different structure is the one of the delivery package. It includes binary and user-documentation files and is adjusted to be most convenient for the delivery (note also that different structures can be defined for different delivery media). This structure is derived from the development structure by a build and release activity. Finally we have a structure on the target system, when the software is installed. For embedded systems this structure is the same structure at run-time, and it is often hidden from users. For non-embedded systems, or large systems, we differ two structures – installation or deployment structure where the software is saved on a media and ready to run, and run-time structure in which the software is executed. In both cases this structure is usually hidden from the user, so we can consider it as one structure.

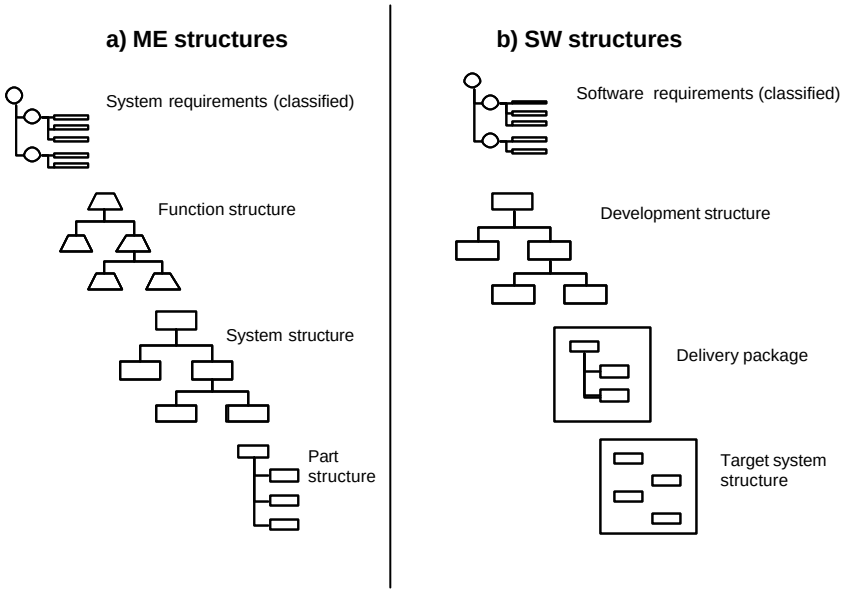

Figure 3. Structured information used in ME and SW development process

Comparing ME and SW structures we can find many similarities. Both start with requirements, overall system design and identification of system functions. The function structure for ME design is connected to a system structure which corresponds to the physical structure of the system. This structure is later related to the part structure where the physical parts of the systems are specified. Software does not have physical components like hardware. For this reason it is possible to use a development structure that corresponds to the system architecture, i.e. to a logical structuring. However, when the software is built, the internal functional structure disappears. This degeneration of the structure may cause problems in the software maintenance. Traceability problem (i.e. the possibility to find the cause of a problem) is one of the main problems in software maintenance.

3. The tools

The main prerequisite for a uniform and well integrated process is the integration of the tools supporting these processes. PDM systems integrated with other tools (such a CAD/CAM tools) are widely used in ME development. Software development uses SCM tools integrated with development environment tools, such as editors, compilers. etc. What does these tools have in common? What information do they process? To answer to these questions, let us look first at these tools.

3.1 PDM systems

PDM systems are computer systems which are used to handle product data. The user functions of a PDM system can be defined as [CIMdata 2001]:

- *Data Vault and Document Management* – Documents must be stored in an organised manner. Information about the documents (meta-data) is stored in a central database. Document management routines are used to manage release and change of documents.
- *Workflow and Process Management* – Routine processes can be monitored and controlled by a PDM system.
- *Product Structure Management* – Product structures are defined and changes of them controlled.
- *Part and Component Management (Classification & Retrieval)* – Standard parts can be classified to support re-use.

– *Project Management* – A large project can be broken down in sub-projects. The progress of a project can be tracked.

When it comes to managing product structures, PDM systems traditionally aim at managing part structures. A parts structure is typically built of assemblies and parts, possible variants of the parts, revisions of them and relationships to documents. Most PDM systems are customisable and it is possible to define specialised object types, such as functions and systems. This kind of object types can be found in some PDM implementations.

PDM systems are not specialised for managing requirements. Requirements management is often supported with specialised systems, called RM (Requirements Management) systems. To achieve good traceability between requirements and product information, and thus support for a product model spanning from requirements to life-cycle systems, as in figure 3a, integration between the PDM system and the RM system is needed [Svensson and Malmqvist 2001]. Integrations between commercial systems will be available in the near future.

3.2 SCM systems

SCM systems are used to control the evolution of complex software. From a management perspective, SCM directs and controls the development of a product by the identification of the product components and their related documents, product structuring, control of their continuous changes, status accounting and auditing. From a developer perspective, SCM maintains current components, stores their history, offers a stable development environment, build the products and co-ordinates simultaneous changes in the product. SCM includes both the product (configuration) and the working mode (methods) and the goal is to make a group of developers as efficient as possible in their common work with the product.

SCM treats first of all complexity of software products. As software became more complex, the SCM functions increased. Today, there are many disciplines covered by SCM, and the most important are:

– *Version and Configuration Control* - The possibility to store different versions of software items to subsequently be able to retrieve them and select particular versions in a configuration.

– *Build and release management* – Mechanisms for keeping generated files up to date, managing final products in form of software packages suitable for distribution and installation.

– *Workspace management* – A support for developers working in a project. Controlling the working versions of the modules being developed.

– *Change management* – A system supporting the management of the collection of change requests, for instance in collaboration with customer support, the generation of error reports, firm change requests, implementation of those changes, documentation of the problem and the solution, and when it is available.

– *SCM Process support* – In the recent years, especially as a consequence of emergence of CMM the process aspects of the SCM activities, i.e. their planning, executing and measuring, became an important part of SCM.

These disciplines are much related to the software development cycle: Editing source code and building it. The products of that process, i.e. binary files, traditionally were not part of SCM, since it was easy to re-produce them. The objects treated by SCM were mostly source code modules.

3.2.1 Comparison of the tools

Even if PDM and SCM are specialised systems with unique functionalities, there are many functions common to the both systems. Table 1 compares the functionalities of the both systems.

PDM systems are strong on product structure management. Moreover product modelling in SCM is very weak. SCM systems do not contain a product model. The differences in approach come from fundamental differences in the nature of hardware products: In PDM, the product has a physical existence and consists of physical parts. For that reason the product structure can be represented by a part structure. In software there is no such physical structure.

The strength of the SCM systems lies in the support for software development, by supporting its specific demands for version management [Persson *et al*. 2001] and by providing functions such as build and release management. However, the software development process is getting more complex. In addition to traditional source code management, some new SCM systems can manage other type of objects, with different internal structures and different types of relations. The previously dominant version management has been overwhelmed by the configuration/selection discipline. In this respect SCM became less software-specific and more similar to PDM solutions.

Finally, it should be noticed that change management is supported by both tools.

Table 1. PDM and SCM functions (Adapted from [Persson *et al.* 2001])

Type of functionality	***PDM***	***SCM***
Version Management	Yes (basic)	Advanced
Product Structure Management	Yes	No
Build Management	No	Yes
Change Management	Yes	Yes
Release Management	Yes	Yes, but weak
Workflow and Process management	Yes	No
Document Management	Yes	Partly
Concurrent Development	No	Yes
Configuration/Selection Management	No	Yes
Workspace Management	No	Yes
Roles	Yes	Yes, but weak

4. Overall process integration

From Table 1 we see that PDM and SCM tools share many common functions. However it is not sufficient to look at the support of the tools, but the overall process must be taken into account. We must look at the possibility of process, information and, finally, tool integration.

4.1 Process integration

To be able to control the development of an ME/SW product, the two processes must be integrated in an overall development process. Figure 4 shows the activities from a typical systems development model [Stevens *et al.* 1998], which describes how a system is developed by a successive division into sub-systems, which are developed in parallel, followed by integration of them. The subsystem developed can be realised in ME, software or any other technology. In Figure 4, the ME and SW design processes are integrated. They run as parallel processes after the architecture has been defined.

To be able to control how the development succeeds in terms of the maturity of the design, life-cycles needs to be defined for relevant information objects, such as systems and components, to define which states the object are in. The maturity of the design, can then be captured by indicating in which life-cycle phase an object is. Life-cycles need to be defined both for SW systems, ME components and the overall system, see Figure 4, which also explains how the life-cycles can be related with the process.

4.2 Information integration

Before going into the discussion of how to integrate the tools, it is important to identify the requirements for common data management. How is the information from the two domains best integrated? We have found two alternatives: (A) Treat the software as components by managing the executables as such or (B) Treat the software's internal structure as a part of the system developed.

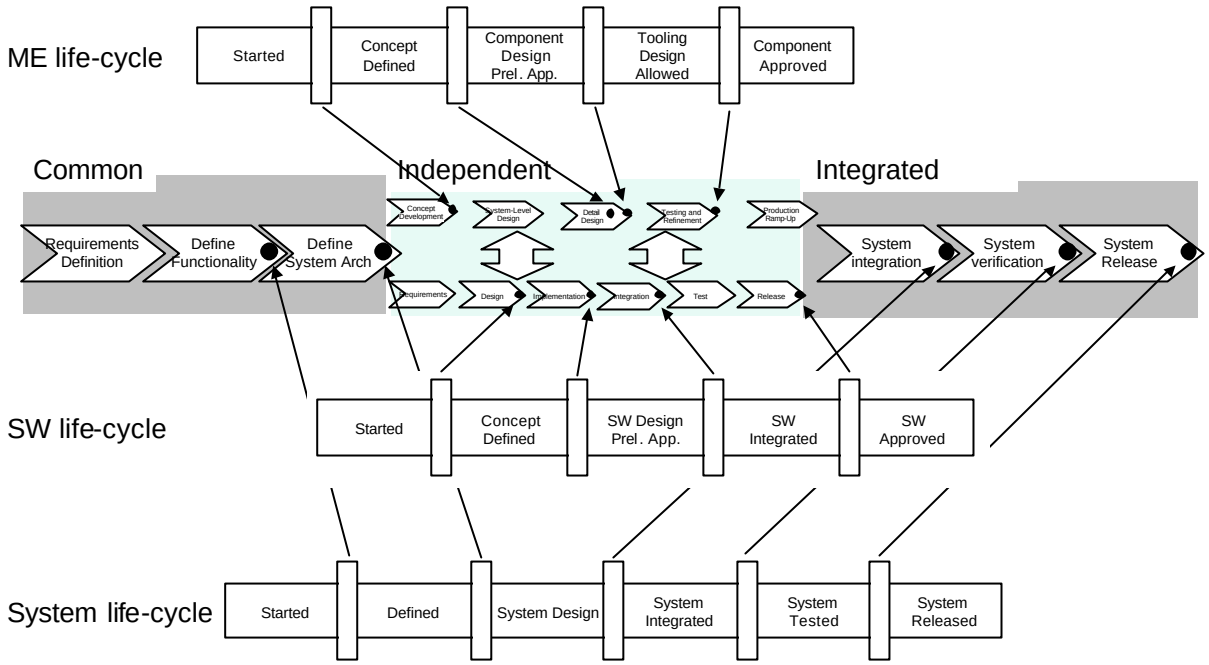

Figure 4. Integrated ME and SW design processes with connection to life-cycle phases

Alternative (A) is a consequence of a hardware oriented view on product development and information management. According to the hardware oriented approach, the product is comprised of components, which are assembled in a manufacturing process. Hence, the software object corresponding to a component is an executable, which is downloaded to the product in the manufacturing process. This way of managing software information is sufficient from a supply point of view, but the design process puts higher demands on information management. One single software system, delivered as one executable or as an installation package, contains many functions in form of applications, packages, data, etc., fulfilling a number of requirements. To achieve traceability between requirements and software sub-systems, a more detailed product model is needed. An analogy to treat software as executable components, would be to manage a car engine as a component during the development process, just because all parts are mounted together.

Alternative (B) takes one step further and manages not only the executables, but also other kinds of structured information related to the software, such as requirements, documents, baselines, and software specific product structures, such as the installation package structure. This way, better traceability from system level and down to components is enabled.

In Figure 4 we see that there are isolated points in the development life-cycle where activities are split up to separate activities and when they are again merged together. At the split point it is important that all system related information created is accessible and that it is possible to import this information into the tools used in the following activities. Figure 1 and Figure 2 show the information flow through the phases. From these figures we can summarise that we need the information flow of the following assets:

From common activities to the parallel ME and SW activities we need requirements, change requests and system design (defined partly by a product structure). From the parallel activities to the integration phase we need requirements, final detailed design and the final deliverables (executable code for software, prototype specifications for ME and documentation for both). From this we can conclude that for both ME and SW development processes should follow common procedures for: product structure management, requirement management, change management and in general document management.

4.3 Tool integration

The basic requirement for an integration is to be able to control the parts and software systems building the product. This includes managing their versions and their combinations.

The system structure is the logical point for integration. The PDM system manages the system structure (as a product structure), which in turn points out how the systems are realised. The ME parts structures are easy to relate to the system structure, since they can be stored in the PDM system. The development structure in the SCM system must be referenced to in some indirect way. From the SCM system, the following objects must be transferred: Source code baselines, executables, and life-cycle status of the SW system developed. The source code baselines can be stored in the PDM system the same way as parts structures, and the life-cycle status of the SW system can be stored as a property of the system object. The executables are often stored in archives and referenced from the PDM system.

This integration has achieved the following: (A) The information about the product with its ME and SW constituents is available. (B) It is possible to follow the maturity of the design for both a system as a whole, and for its ME and SW constituents. This facilitates the management of the complete product, both from an information management perspective and from a project management perspective.

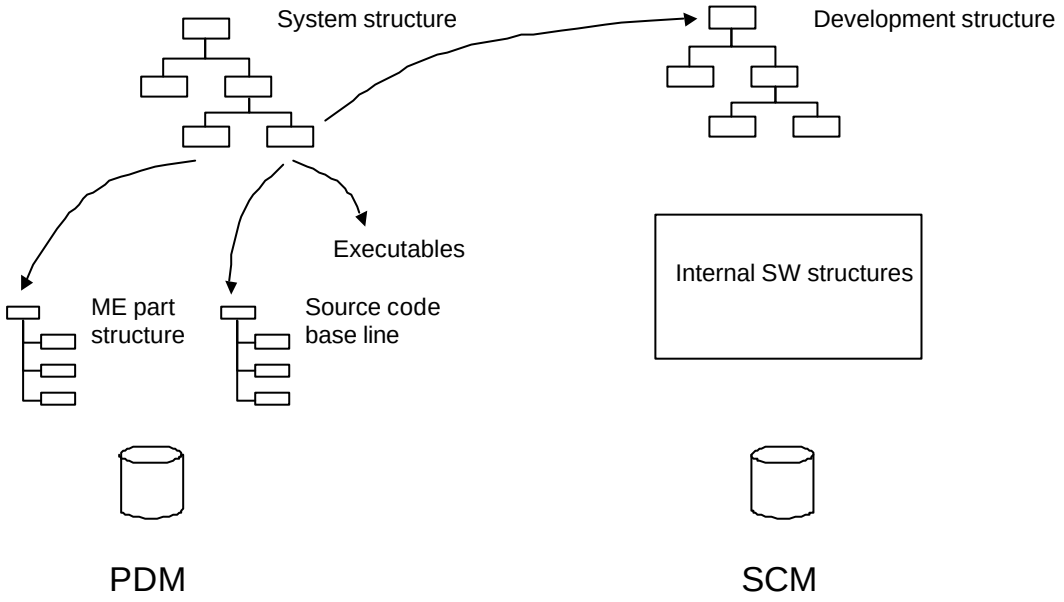

Figure 5. Integration of PDM and SCM systems

The previous description roughly outlines the information system architecture for the integration, with emphasis on product structure management. Further decisions have to be made about how to best utilise the functionality of each system. As we pointed out in the previous section, requirements management and change management are common activities for ME and SW design. However, we consider product structure management to be an important prerequisite to be able to successfully perform those activities.

5. Conclusions and future work

In order to effectively manage ME and SW development, a systems development perspective is needed. In the early phases, before the technology is selected, the development process can be treated as a common process, while the process splits in two paths during the development of the ME and SW sub-systems, which meet again at integration time. The system is the common concept for ME and SW design. Therefore, to control the evolution of the system developed, a life-cycle must be defined, not

only on component level, but also on SW level and overall system level. To be able to manage the information for ME and SW together, requirements, functions and systems are objects common for ME and SW development. Below system level, different information is used in the two domains. A PDM system can manage system structures for the complete system, and also component information for the ME parts of the system. An SCM system can manage the internal structure of the software developed, including the detailed version management of source code files. Executables and the relationships between system versions and software versions can be managed at the system level in the PDM system.

An interesting subject for future research is to study the requirements on information management from other perspectives than design, with manufacturing and maintenance as those most important. Further, the information management in the ME and SW domains must be compared on a deeper level. To be able to have an integrated information management, a common terminology and understanding between the two domains must be achieved.

References

Andreasen, M. M., Hein, L., " Integrated Product Development", IFS Publications Ltd., London, 1997.

CIMdata, "collaborative Product Definition management: An Overview", CIMdata Inc., Ann Arbor, MI, USA, 2001.

Crnkovic I., Persson Dahlqvist A., Svensson D., "Managing Complex Systems – Challenges for PDM and SCM"

IEEE Proceedings, Asia-Pacific Conference on Quality Software Hong Kong , December 2001.

Estublier J., Favre J.-M., Morat P., "Toward SCM/PDM Integration?", System Configuration Management, SCM-8, Lecture Notes in Computer Science 1439, Springer, pp. 75-94, 1998.

Persson Dahlqvist, A., Asklund, U., Crnkovic, I., Svensson, D.,Ranby, J., Hedin, A., Larsson, M., "Product Data Management and Software Configuration Management: Similarities and Differences", Technical Report, The Association of Swedish Engineering Industries, ISSN 1493-6444, 2001.

Sommerville, I., "Software Engineering", Sixth Edition, Addison-Wesley, Harlow, UK, 2001.

Stevens, R., Brook, P., Jackson, K., Arnold, S., "Systems Engineering - Coping with Complexity", Prentice Hall, Hertfordshire, UK, 1998.

Sutinen, K., Almefelt, L., Malmqvist, J., "Implementation of requirements traceability in system engineering tools", Proceedings Produktmodeller´2000, November 7-8, Linköping, Sweden, 2000.

Svensson, D., Malmqvist, J., "Integration of Requirement Management and Product Data Management Systems", Proceedings of DETC'01, Paper No DETC2001/CIE-21246, Pittsburgh, Pennsylvania, USA, 2001.

Ulrich, K. T., Eppinger, S. D., "Product Design and Development", McGraw-Hill, Singapore, 1995.

Daniel Svensson, PhD Student
Chalmers University of Technology, Product and Production Development
SE - 412 96 Göteborg, Sweden
Telephone: +46 31 772 5855
Telefax: +46 31 772 1375
e-mail: daniel.svensson@me.chalmers.se