
Managing near decomposability in complex platform development projects

Ramsin Yakob* and Fredrik Tell

Department of Management and Engineering,
Linköping University,
SE-58183, Linköping, Sweden
E-mail: ramsin.yakob@liu.se
E-mail: fredrik.tell@liu.se
*Corresponding author

Abstract: Product Platforms are understood as the sharing of common and underlying technological assets from which a number of derivative products can be produced. The large number and character of technological interdependences give rise to system complexities, implying that product platforms can be managed as nearly decomposable system. We study the managerial implications of such complexity in a telecommunications platform enhancement project. We find that managing near decomposability requires the ability to provide feed-back and feed-forward information, relating to the implementation and planning activities of such a project and that this required output can be provided through the process of aggregation.

Keywords: product platforms; R&D management; project management; near decomposability; organisational learning.

Reference to this paper should be made as follows: Yakob, R. and Tell, F. (2007) 'Managing near decomposability in complex platform development projects', *Int. J. Technology Intelligence and Planning*, Vol. 3, No. 4, pp.387–407.

Biographical notes: Ramsin Yakob was awarded his BSc from the University of North London, UK, and his MSc from School of Business, Economics and Law, Göteborg, Sweden, both within the field of International Business. He has worked as a consultant with the Royal Bank of Scotland, UK, and at American Express, UK. He currently is a doctoral candidate at Linköping University, Sweden. His dissertational research areas are platform technologies, knowledge management, and project organisation and innovations.

Fredrik Tell was awarded his BSc, Lic Econ and PhD from Linköping University, Sweden. He currently is an Associate Professor at the Department of Management and Engineering at Linköping University. He has also been a Lecturer at the London School of Economics and Political Science as well as a visiting research fellow at University of Sussex and Stanford University. His main research interests includes innovation and industry dynamics, management of complex technologies, knowledge management, history of technology and business history. He has published papers and reviews in journals such as *Business History*, *Industrial and Corporate Change*, *Management Learning*, *Organization*, *Organization Studies*, *Research Policy*, and *Technology Analysis and Strategic Management*.

1 Introduction

The platform concept has received increased attention in recent years. It has become an imperative strategic feature in many organisations and in a range of industries (Gawer and Cusumano, 2002). Consequently, the deployment of platforms can be found in many different business spheres of a company. It is today feasible to speak with reference to the existence of product platforms, process platforms, customer platforms, service platforms, brand platforms, and even global platforms in many companies (see Halmann et al., 2003; Meyer and DeTore, 2001; Sawhney, 1998). It has been argued that a platform approach to product development and manufacturing carries numerous benefits since it supports and facilitates auxiliary product development, reduces development, production and investment costs, and augments product and strategic flexibility (e.g., Krishnan and Gupta, 2001, Muffatto and Roveda, 2000, Wheelwright and Clark, 1992).

Although the mounting use of the platform concept across numerous dimensions of the firm is still most commonly associated with the development of new products, subsequently many firms employ or are considering employing product platforms as part of their new product development process (Krishnan and Gupta, 2001). A product platform is defined as the underlying technological foundation made up of a set of assets, components, sub-systems and interfaces purposely planned and developed to form a common structure from which a stream of derivative products can be resourcefully developed and produced (Meyer, 1997, Meyer and Lehnerd, 1997, Robertson and Ulrich, 1998, Krishnan and Gupta, 2001, Nelson et al., 2001, Muffatto and Roveda, 2000).

Product platforms are strongly affiliated with the concepts of product derivatives and product families. A hierarchical divergence can be used to distinguish between these three categories. The platform is usually created in an early project in which the platform and the initial platform product embodying other non-platform components for commercialisation is created. Derivative products are individual products within a product family and product families can be understood as the attempt of companies to generate a continuous stream of products based on mutual common technology, often a platform (Meyer and Lehnerd, 1997). Further, the process of developing platforms and derivative products is not the same and is believed to differ on a number of issues, one being with reference to technology (Halman et al., 2003, Tatikonda, 1999).

One such difference can be found at the level of technological interdependence, represented by the extent to which components, sub-systems, systems, and processes-related technologies interact and affect each other (Tatikonda, 1999). Technological interdependence is of importance to platform development since platforms can be understood as comprising two major components, its major sub-systems and the interfaces between these sub-systems (Meyer and DeTore, 2001). This in turn means that platform development often takes place at the level of specific sub-systems (defined as a logical unit of technology delivering a specific functionality required for the overall system) and at its interface points (defined as the connection between sub-systems within a platform). Since major sub-systems themselves can have their own specific sub-systems, a platform like so many other complex products consists of an architecture (Henderson and Clark, 1990; Hobday, 1998). This architecture of the platform defines the functional requirements within the system, maps its requirements to physical elements or sub-systems, and describes the interaction between these physical elements (Yassine and Wissmann, 2004, Ulrich and Eppinger, 1995, Ulrich, 1995).

Thus, in projects dealing with the development of large technical systems, a high degree of complexity can be expected. In such systems, the number of components and sub-systems, the number and nature of interdependencies and interfaces, and the way these are hierarchically configured, contribute to their complexity. Simon defines a complex system as:

“one made up of a large number of parts that interact in a non-simple way...//...given the properties of the parts and the laws of their interaction, it is not a trivial matter to infer the properties of the whole.” (Simon, 1962, p.468)

This paper aims at remedying two limitations in existing literature on platforms in product development processes. First, platform focused research has often investigated products that are a construct of a platform approach to development and manufacturing, a platform-based product development approach (see e.g., Krishnan and Gupta, 2001, Muffatto and Roveda, 2000, Nelson et al., 2001, Meyer and Lehnerd, 1997, Dawidson et al., 2004). Research relating the development of the underlying product platform itself and its processual and managerial implications is much more limited. Second, while previous research has indicated the systemic aspects of product platforms, the complexities arising are suggested to be primarily dealt with by using modular approaches (see e.g., Ethiraj and Levinthal, 2004, Baldwin and Clark, 2000, Ericsson and Erixon, 1999). This suggests that the conception of complexity arising from using platform concepts is one of analysable complexity (Lindkvist et al., 1998) or perfect decomposability (Marengo et al, 2005; Simon, 1999). Our findings from a case study of a telecommunication platform development project suggest otherwise, namely that complex platforms imply nearly decomposable systems, and that the management of such processes call for alternative approaches to managing product development projects than those in extant literature. Further to this, there is little systematic research undertaken of how to partition a system and what the inherent risks of partitioning wrongly are (Ethiraj and Levinthal, 2004). We argue that in-depth studies are required to improve our knowledge of project management in complex product development. In particular, we are concerned with the managerial and processual implications of managing near decomposability as we believe that this carries specific implications for the management of complex projects.

This paper is structured as follows. Section 2 introduces the discussion on near decomposability. Section 3 briefly discusses the methodological approach taken. In Section 4, we provide the case illustration of a platform enhancement project in a firm in the telecommunications industry. We conclude this paper by discussing decomposition and composition and elaborate on the importance of feedback and feedforward as two important learning mechanisms.

2 Understanding product platforms as nearly decomposable systems

The measure of system decomposability is a function of number, degree, and level of interaction between sub-systems within a system. When there is no interaction between sub-systems, the system is fully decomposable. Where some interactions are shared by all parts in a system and the interaction taking place within sub-systems are more frequent and tighter than it is between sub-systems, a system is rendered nearly decomposable (Simon, 1962).

Whereas perfect decomposability has been described as the isolation of components and problems into optimised and independent entities, which can be developed and solved independently, near decomposability implies that such a perfect decomposition is not possible. The overwhelming majority of complex problems or systems are not perfectly decomposable but nearly decomposable (Marengo et al., 2005, Brusoni et al., 2004). The difficulty of decomposing a complex system into partitioned, independent parts, or modules, gives rise to sub-optimisation issues. What near decomposability acknowledges is the existence of a high number of elements, interdependencies, and interfaces that cannot and do not need to be decomposed to a level of independent units. It implies the decomposability of complex systems and problems to a degree where only the most relevant interdependencies are contained within each component, less relevant ones can persist across sub-problems (Frenken et al., 1999; Simon, 1996). A nearly decomposable system has been illustrated ‘as a boxes-within-boxes hierarchy with an arbitrary number of levels’ (Simon, 2002, p.589) and emphasises the near-independence of system components.

“(1) In a nearly decomposable system the short-run behaviour of each component sub-system is approximately independent of the short run behaviour of the other components; (2) in the long run the behaviour of components depends in only an aggregate way on the behaviour of the other components.” (Simon, 1996, p.198)

According to a number of authors, the issue of near decomposability is a matter of problem-solving strategy (Marengo et al., 2005, Frenken et al., 1999; Brusoni et al., 2004; Simon, 1996). Problem-solvers are often forced, due to their limited computational capacity, to decompose problems, which are not fully decomposable. Marengo et al. (2005) discuss problem-solving through the coordination of a large number of interacting and interdependent entities, which altogether contribute to forming a solution to the problem itself. According to these authors, difficulty arises as a result of

“opacity of single entities’ functional relations and the partial understanding of their context-dependent individual contributions in forming a solution to the problem at hand.” (Marengo et al., 2005, p.2)

Near decomposability can thus be understood as the difficulties of anticipating consequences on the development task as a whole arising from a particular action. Clark (1985) argues that system design and system architecture is a search for the understanding of what the object or product is and ought to be, given the context in which it must function:

“the working out of a design involves a process of analysis, of identifying the components of the form, the major systems and sub systems, and then grouping them in different ways to illuminate their interrelations.” (Clark, 1985, p.241)

Clark (1985) states that given the nature of uncertainty inherent in complex objects, interrelations among aspects of the product are unlikely to be fully understood and contradictions unlikely to be resolved at the initial conception of a design. While decomposing a problem is necessary in order to reduce the dimension of the search space, it also shapes and constrains a search process to a specific sub-space of possible solutions (Marengo et al., 2005). Consequently, optimal solutions cannot be generated as systems are locked into sub-optimal solutions. By separating these interdependent elements into sub-problems, complexity can be reduced to smaller more manageable

sub-problems. In this way, optimising each sub-problem independently will not necessarily lead to overall optimisation, but to one of 'satisfying' solutions (Marengo et al., 2005; Brusoni et al., 2004).

Another salient characteristic of a nearly decomposable system concerns the rate of interaction within and between components and sub-systems, interaction predominantly taking place through interfaces. Higher frequency and intensity of interaction takes place between components belonging to a single sub-system than between components belonging to different sub-systems, and this principle holds for all levels of the hierarchy. When responding to changes and input, the immediate effect of such a response is easier made at a lower level of any system. This also means that at the lower level, sense of parity and equilibrium with other parts at the same system level is more rapidly achieved (Simon, 1999).

However, since there will be a systemic effect of changes made in it, it also becomes important to understand total system effects. Each sub-system part affects the total system at the same time as the total system affects each sub-system part. In a nearly decomposable system, each component or sub-system at one hierarchical level determines the development of the next set of components or sub-systems in a higher hierarchical level. Although it is possible to break down a system into its constituent parts, establishing the aggregate effect each part will have on the whole system is riddled with difficulty. The dynamic effects of design decisions for a higher hierarchical level are slower than that of lower levels, making them more difficult to identify. The complexity of a system is related to its degree of decomposability, which in turn depends on the systems architecture. Complexity indicators should, therefore, relate to the decomposability of the specific products architecture (Frenken et al., 1999).

With regard to product architectures, a broad distinction between modular and integral architectures has been made (Ulrich and Eppinger, 1995). In a modular architecture, the interface rules between components and sub-systems is simpler to establish and the mapping of relationships between these can be done with less effort than in integral ones. In an integral architecture, interface rules are less established and the mapping of component and sub-system relationships and interfaces are both complex. A nearly decomposable system consists of both modular and integral aspects of systems architecture. The difficulty aspect is to distinguish between what is or can be modular and what is or needs to remain integral (Shibata and Yano, 2002). In an integral architecture, all component interdependencies and interfaces cannot be easily established and consequently the affect on the system on a higher level cannot either be established. The development approach must provide for those instances where interdependencies and interfaces can be established and for those instances where the aggregated effects cannot. To end up with an integral architecture, one needs to either plan for it from the beginning, or go through a conversion process that may involve re-architecting or rewriting some components (Sun Microsystems).

3 Methodology

The empirical foundation of this paper is a single case study of a successful platform enhancement project in a leading global telecommunications company. The project studied was complex in terms of time, size, organisational design, and technology. It ran over a time period of 24 months, involved over 200 engineers, contained complex

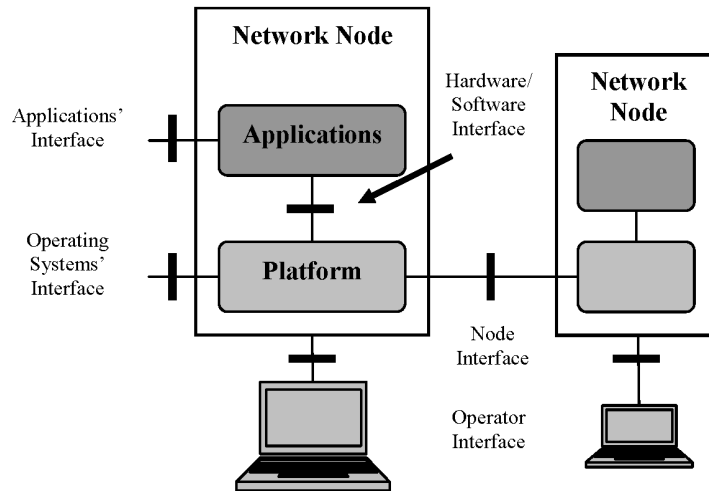
software and hardware configurations, and had development activities taking place in a number of geographically distributed locations. The research method applied in this study is an amalgamation of an interpretive single case study (Walsham, 1995, 2006) and a systemic combining approach (Dubois and Gadde, 2002). Accordingly, we aim to provide deep revelatory representation of the case studied and hope to generalise our findings through the development of concepts, drawing on specific implications or contribution of rich insights (Walsham, 1995). Empirical data gathering was initiated approximately one month after product release and was carried out during the period May 2006 to November 2006. Data and information was collected from three sources; interviews, internal material, and existing research. Interviews conducted lasted for approximately 90–120 min and a semi-structured interview approach was used. Thirteen interviews were carried out with project members in different hierarchical positions. Throughout the research process, our understanding of the phenomenon studied and our theoretical considerations converged through the iterative comparison and contrast being made between extant literature reviews and conceptual understanding.

4 The case study

The company studied is a leading global telecommunications firm, which develops, produces, supplies, and installs telecommunications systems. One of its major business areas is in the provision of telecommunication platforms. A telecommunications platform is a software heavy platform, which also consists of physical attributes. Often, it is the software that determines telecommunication platform functionality whereas the hardware determines its capacity. In its simplest form, it can be understood as a system on which other programs or operating systems operate. It denotes a specific technology and architecture consisting of content and user application layer, a communications control layer, and a connectivity layer composed of hardware, an operating system, and software applications on top (Figure 1). It furthermore serves as a technological and architectural base for the development and evolution of additional systems and applications. At the completion of a platform enhancement project, the platform product is regularly released as a newer version of an existing product, i.e., version 1.0, 1.1, 2.0 ... 6.0 and so forth. In this paper, we will use the term platform to denote the telecommunications platform.

Operating in a dynamic and competitive industry, the company is challenged to continually enhance its existing platform base to stay abreast of new customer and market demands and to defend and increase its market shares. Thus, the company operates several platform enhancement projects in parallel, projects that generally run for several years. One such project was initiated based on a decision in 2003, following a three month feasibility study. The aim of the project would be to introduce an enhanced version of an existing post-3G telecommunications platform. Project goals were to deliver the enhanced platform on time, at specified cost, quality and with functionality according to the main requirements specification. An upgraded platform was required to make new deliveries as well as facilitating the upgrade of a large and already installed base. Important criteria were the provision of reliable and stable nodes, the ability to adhere to delivery dates, cost efficiency and low manufacturing costs. Ensuring development work quality was thus very important.

Figure 1 Telecommunications platform



However in 2004, at the initiation of the platform enhancement project, adherence to quality and delivery precision requirements had become a concern for many.

“Design quality was low when we delivered to other units and we had poor delivery precision. When we said that we were going to deliver at one point it never happened. Or it did but then we had to make a number of error corrections later on. So in essence the project was delayed anyways.”

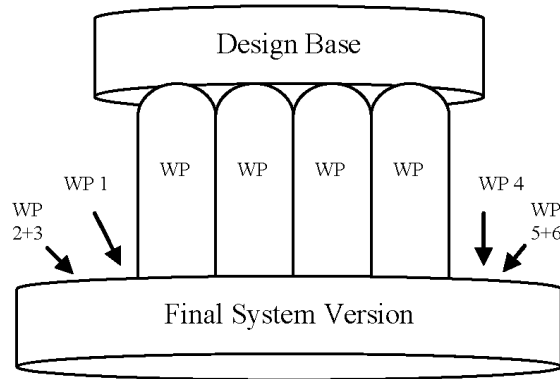
(Assistant Project Manager)

Consequently, top management decided to address the structure of the product development process. In conjunction with this, increased complexity demands forced management to reconsider the traditional project model and introduce a new approach towards the way the enhancement project was organised and managed.

“The reason for introducing a new way of working and a new build environment was a result of panic of not having the right quality. That required proper and drastical measures because we all knew that we could not continue in this direction but that something needed to be done.”

(Assistant Project Manager)

Attempting to reduce costs, increase quality and flexibility, and shorten delivery time, the firm augmented its experience and learning from previous platform development and enhancement projects. The project model used in previous platform enhancement projects (Figure 2) can be best explained as concurrent engineering. The figure illustrates the initiation with a common design base on which larger work packages were created and developed in concurrent fashion. Development responsibility was functionally divided and unverified code was delivered and integrated at one point, at the point of final system version creation. This often led to an integration big-bang since larger increments had to be managed. Responsibility for verification and functional tests resided with each product area integration and verification (I and V) unit. Functional verification took place after final systems integration, which meant that errors in functionality delivery disseminated into the work of other development teams and functions. Control of what was working and what was faulty was hard to distinguish.

Figure 2 Previous way of working

The new way of working (Figure 3) was an evolution of the way in which development activities had traditionally been managed within the company and built on a development logic in which the system was developed in smaller but more frequent steps. From a historically sequential approach towards one focused on concurrent engineering, the current approach would allow for a mix of approaches. The figure indicates increased focus on more, frequent and tighter technical integration of development output. Integration Centred Engineering (ICE) would play a focal role. By increased focus on technical integration, it was believed that delivery precision, quality, and flexibility could be increased. Smaller and more numerous work packages would gradually be integrated to form latest system versions, which in effect became the design base for the following set of work packages, leading up to final system version.

4.1 Anatomical decomposition

One of the most important aspects in the early phase of the project was the creation of the anatomy and the mapping of technical development paths. The anatomy (Figure 4) was a visualisation of project work broken down into Work Packages (WPs), their interdependencies and the grouping of WPs into Latest Systems Version (LSV).

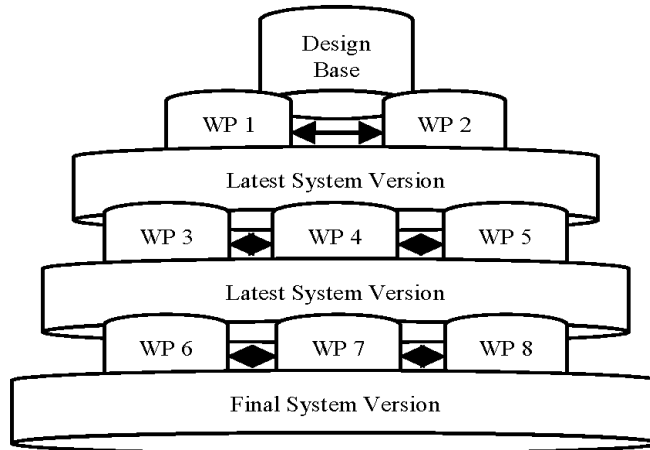
“The idea is to have a holistic picture on paper, and also have the possibility to restructure within the project, when required to.”

(Test Coordinator)

A WP was a functional entity that could be integrated into the system on its own and defined a small addition of verifiable system qualities. Integration of numerous WPs resulted in the creation of a new LSV. A shipment was a delivery (of a LSV) to a customer. The anatomy was a visualisation of dependencies between WPs, dependencies that constrained their order of integration and verification into LSVs. It also visualised internal dependencies between WPs and sub-systems within the system. As such, it provided means to which it was possible to map WPs within sub-systems that required attention first in the development work. The anatomy was also used to visualise the number of shipments planned for the project, their shipment dates, and their content in terms of included WPs. Hence, the anatomy visualised WPs that could be

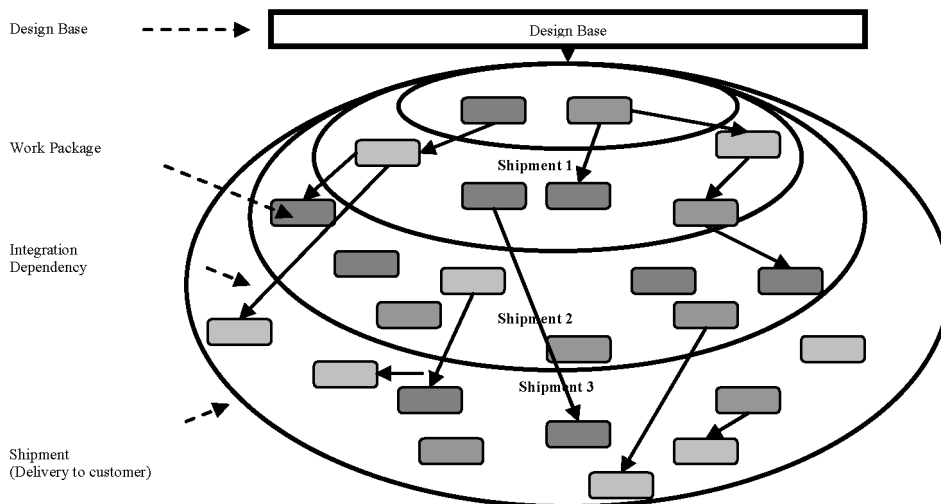
developed and integrated in parallel, which needed to be developed and integrated sequentially and the development steps to be taken within the project.

Figure 3 The new way of working



The creation of the initial anatomy was a demanding task that took approximately three months to complete. The new way of working required more issues than before to be taken into consideration and planned beforehand. The project required planning so that each development step taken was continuous and dependent on a previous step. This would minimise the need for rework. This also meant that the efficiency of the project was highly dependent on each development step being functionally and technically sound. System technicalities had to be planned early on and the creation and quality of the anatomy set the path for the success of the project.

Figure 4 The anatomy



A central part of the process of creating the anatomy was to make all involved sub-projects (sub-system areas) to consider what their task was. Establishing these tasks consequently led to the division of work into WPs. A WP clearly demarcated the functionality that was to be developed and to undergo rigorous testing before delivery for integration. Demarcating functionality was the responsibility of each WP team and these teams were established prior to responsibility being assigned for development work. Creating the WPs and the functionality that should be included in each of them also required some ground-level planning to take place. This was dependent on resource allocation requirements and availability as well as on existing interdependencies between WPs. Despite the encapsulation of specific functionality and the early establishment of interdependencies, the acknowledgement of existing interlinkages between WPs required the participation of a large number of people in the creation of the anatomy. In addition to this, the inherent complexity in mapping the system from start to end and due to system size, the number of system areas involved, and the number of dependencies, meant that dependencies were too many and too complex to be established by one single individual.

“At a lower sub system level it was possible for a single person to establish these dependencies but at the level of total system it still required a group of knowledgeable people to make the feasibility study and the initial Anatomy”.

(Assistant Project Manager)

With the creation of the anatomy, the project was planned from start to end. However, project management had recognised the difficulties of planning too far ahead into the future on a detailed level. Hence, there was a recognised difficulty to establish all dependencies and interdependencies at the outset of the project and in the Anatomy creating process.

“It is important to have a good understanding of the interdependencies between the different work packages. The reason is that a newly identified interdependency later on in the project could mean that the whole project needs to be re-planned. And that is the real vulnerability in this process. If you don't succeed in, and have the preconditions to create the Anatomy then you will have trouble later on.”

(Assistant Project Manager)

Consequently, re-planning the anatomy at several points became necessary. Thus, the anatomy had to provide enough robustness for the work to be carried out and the project to progress yet maintain sufficient flexibility to accommodate changes. Reorganisation of WPs had to be made, taking into consideration the interdependences that already existed. Some WPs were consequently delayed whereas others were put forward. Discussion and decisions concerning changes to the anatomy took place a couple of times a year, often in various meetings and technical forums.

These meetings were important to obtain full technical control, coordination, and information sharing in the complete project. At these technical forums, interdependent aspects of development work and cross-WP impact of integration were discussed. They provided an opportunity for project representatives to run through the anatomy and discuss its evolution, any delays, and also required changes. Though they fulfilled an important role in bringing together representatives from all sub-projects and facilitated information sharing, the actuality of establishing future integration impact was still limited. As a result, the anatomy was considered highly dynamic, being updated and

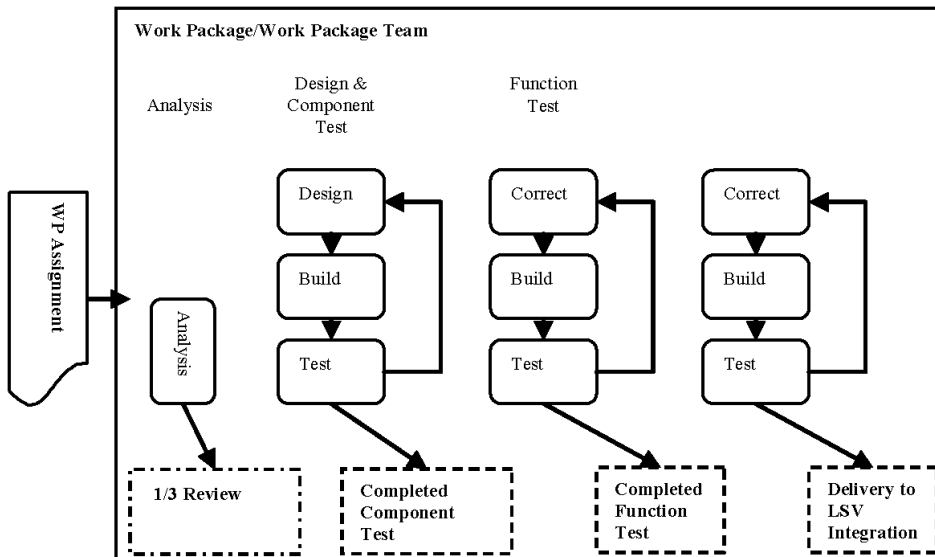
restructured whenever new dependencies and interdependencies were identified. By enabling the re-planning of the anatomy, new requirements, both internal and external, could be included in the system development work.

4.2 Encapsulation

All functionality to be developed for the system was encapsulated into WPs. Through the creation of WPs, it was possible to divide the development work into small integratable and verifiable parts, which further established the order in which functionality should be developed and integrated into the system. Within a WP, end-to-end responsibility for functionality development, including analysis, development, test, and verification were included (Figure 5).

Further to this, a WP should always be sufficiently complete for integration into the system and still ensure that the system worked. This required that each WP would be verifiable and backwards compatible. Compatibility would allow integration into the system without interference with the functionality of the existing design base or previously delivered WPs. Parallel development and parallel integration with other WPs was another important characteristic of a WP. Finally, a WP was advised to be sufficiently small so that it could be totally removed if necessary due to changing prioritisation or resource shortages. One major advantage of working with WPs was that it was flexible enough for new requirements to be added throughout the project. It allowed WPs to be split if they become too complex and contained too much functionality. These newly created WPs could then be included somewhere else in the anatomy. This was different from running larger increments where any such new large increment would have an effect on the whole structure, and hence require a different kind of project re-planning process.

Figure 5 Work package



“What is important is that these work packages have their own identity. That is important in order to be able to make changes in the Anatomy.”

(Assistant Project Manager)

Development teams were assigned to each WP. A WP Team was a cross-functional development team responsible for carrying out development work, consisting of employees from organisation and product areas such as system, design, test, and integration. Development work was carried out in a local system environment, separated from all other development work. This means that a WP team was always operating on their own copy of the LSV, referred to as the Latest Local Version (LLV). This ensured ‘local’ system testing and verification before making delivery for integration into the LSV. The reason was that to avoid that ongoing work within one WP team would impact any other WP team or the LSV. The LSV should only contain verified and functional WPs. Thus, increased attention had to be given to testing and verification activities prior to the more frequent integration exercise. Ideally, there would be no requirement for making any additional test or integration between WPs before they were sent to systems integration and verification. However, because it proved impossible to establish all dependencies at the start of the project, there was a need at times to ensure that two work packages could be integrated together at system level.

“You need to be aware of the interdependencies on a deep technical level, as deep as software module level, and that is a pretty tough task in a project of this size.”

(Assistant Project Manager)

4.3 *System aggregation*

A central part of the new way of working was the integration of WPs into a periodically created aggregated whole (LSVs). This was an important activity that ran throughout the project from its initial point up until final system verification.

“Integration happened at several levels in the project. It happened within work packages, between work packages and then it happened at System Integration and Verification. It was more of a fluid process where the strict hierarchical division was abandoned.”

(Technical Coordinator)

An LSV was a functioning part of the anticipated final system product, created every second week. For each new LSV produced, more functionality and characteristics would be introduced in the system. Because an LSV contained working functionality, it could be delivered to customers. Within the project, five LSV shipments had been planned to be carried out and delivered to customers before the final product system was finished.

“The only difference between a shipment and the final system, apart from that quality and functionality is improved, is that no new additional functionality has been added. In essence each shipment is a product that can be seen as a functioning system to be used, although it has a limited capacity in comparison with the final product.”

(Test Coordinator)

Since integration issue was the key determinant for delays, as had been shown historically, an approach whereby gradual integration could take place was required. This carried effects for not only the internal development process but also influenced internal customers concerned with developing platform applications.

“Breaking up system delivery to customers in five different shipments was decided in order to avoid too much functionality being delivered at the same time, hence having a ‘big bang’ effect’ [on the customer].”

(Technical Coordinator)

It also carried the purpose of facilitating for internal customers to integrate the development output with their own systems. By enabling system aggregation several times during the life of the project, it was possible to facilitate greater and closer integration with immediate internal customers of the product. Such dependencies predominantly came from the need of integrating applications development work in the planning of the anatomy. By doing so, it was possible to establish which functions in the application development work would address which functions in the platform development work and vice versa. By allowing internal customers to gradually integrate the platform with their own development work, parallel development between projects was facilitated.

“... our Anatomy is governed with what parts, which work packages they (Applications) are dependent on in order to carry out their own development. Now they can start their own development after our first delivery because what is most critical for them is included in that.”

(Technical Coordinator)

The process of more frequent system aggregation in the new way of working influenced the development steps taken within the system. Since the integrity of the LSV had to be maintained at all costs, no deviations in accepting only fully tested and verified WPs was made. In instances where there were interdependencies between WPs, joint verification in terms of function tests and regression test were carried out between the WP teams before delivery and integration into the LSV. WPs were small in terms of added functionality, thereby facilitating and speeding up the integration process. However, since several weeks could pass before a delayed WP could be slotted in for integration, these WPs could become very large and numerous. At times, several large WPs were delivered simultaneously, making the integration activity difficult. Integration difficulties were, however, not only taking place at the Integration and Verification (I and V) Unit. Another issue was a consequence of the frequent aggregation of LSV, which would make up the latest design base for WP development.

“What has taken time is to lift up the code base (from LSV) and run your local test with your development before delivery. We haven’t managed to speed up that process.”

(Technical Coordinator)

Final integration of WPs into LSV and the final system version was carried out by the Integration and Verification Unit, which made tests and verification for each WP delivery. I and V integrated all WP deliveries into a new LSV, which then was made public for following WP Teams and consequently became the new (LLV) of the system on which further development work was carried out and verified. Each new LSV was

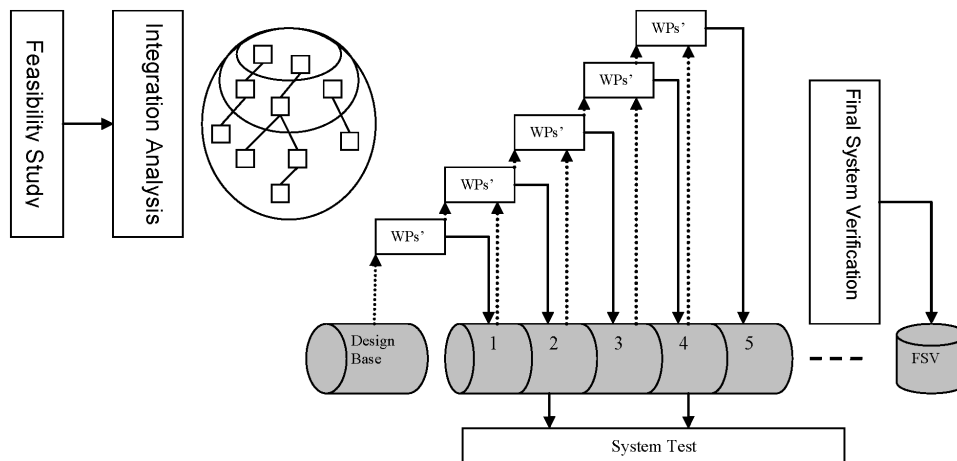
expected to undergo extensive regression test (backward compatibility) by both the central System and Integration unit as well as by each product area's systems integration. The process also ensured that all new WPs that followed the integration and creation of an LSV were working on a similar and the latest design base. When the last WP within a shipment had been delivered and integrated into the LSV and the LSV had been regression tested, the LSV (could) become a shipment that was delivered to customers. When the last shipment or the last function delivery had been produced, final systems verification was performed and the system thoroughly regression tested.

“The complete and exact impact of future LSV's has not been simple to understand until the work has actually been done.”

(Sub Project Leader)

Consequently, it was recognised that planning interdependencies and establishing overall system performance was difficult to establish too far ahead in the future. After approximately two years of development work, the final system version was delivered for commercial use on several markets. The way of working (Figure 6) proved successful in reaching the time and quality criteria set and has continued to be used within the company.

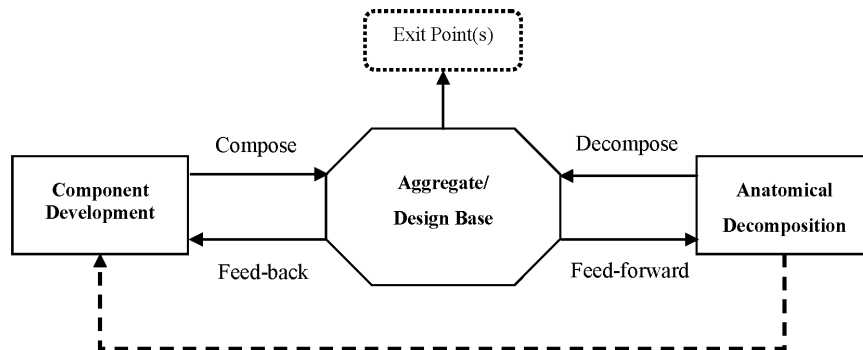
Figure 6 Project model development pattern



5 Managing near decomposability

An approach to the management of near decomposability in complex product platforms is depicted in the model below (Figure 7) and will be discussed in more detail in the following sections. More specifically, the model illustrates the role of system aggregation in the creation of feedback and feedforward informational output and the use of this information for the subsequent creation of the anatomical decomposition. This anatomical decomposition in turn influences component development in the following iterative cycles of the development process. The model also illustrates how feedback and feedforward information influences the ability to compare development quality and contributes to the increased understanding of future platform decompositions.

Figure 7 Conceptual framework: management of complex platform development projects



5.1 Anatomical decomposition and feedback

The ability to visualise a complex problem and the understanding of it is a powerful tool in solving complex problems (Frenken et al., 1999). Similarly, the visualisation of the epistatic relationship between system elements through the creation of the ‘anatomical decomposition’ facilitates the comprehension of the system development process.

Such visualisation can aid in increasing projects members ability to follow overall evolution of the platform development process and can act as a tool that allows project members to stay abreast of, and comprehend upcoming incremental development steps, identify important integration points, and to better understand how their development contribution relates to other sub-systems and the complete platform. Such an ‘anatomical overview’ further provides an organisational decomposition, effectively tying resource needs to system parts. Furthermore, an anatomical decomposition can be used to compare outcomes to expectations, often through a feedback process (Carroll et al., 2002). Feedback information is crucial for the positive or negative reinforcement of prior choices made and attributes to experiential learning (Levitt and March, 1988) and experiential learning itself offers a form of backward-looking wisdom (Gavetti and Levinthal, 2000). Feedback information is related to the technical solutions developed, implemented and integrated into the system. Such feedback on developed components is useful since it allows a comparison with plans to be made and deviations from it to be identified. It also aids in identifying potential faults since it allows quality evaluations to be made and improvement requirements to be established. Feedback information aids in the identification and determination of parts that are impeding on overall platform performance. Rapidly gaining feedback on system performance through the integration of a number of system components is an important characteristic of a flexible development process (MacCormack et al., 2001).

This is important to establish since technical solutions might be accurate at individual component level but still be faulty at an aggregated level since their overall impact might influence aggregated system performance negatively. Yet, the establishment of the end-to-end development requirements of a complex and nearly decomposable system is more than merely a planning exercise. Relying entirely on pre-determined plans is most likely to not only lead to sub-optimisation issues but furthermore limit the ability to incorporate flexibility in the development process. Despite the positive contributions gained by early establishing an ‘anatomical decomposition’, such an exercise is inherently limited in its ability to provide an overall and accurate system architecture,

understanding of platform performance and complete work breakdown structure. Hence, the creation of mental models and plans can contribute extensively to successful project execution but cannot cater for all complexity involved. The near decomposability of a complex system such as a product platform also carries implications for the planning exercise. Whereas feedback information is valuable for making an evaluation to already planned activities, it does not provide extant value as to future directions. For this purpose, a different sort of output is required, feedforward information.

5.2 Composition and feedforward

To create the information structure of a complete system requires a high level of architectural knowledge about how system parts interact and function (Sanchez and Mahoney, 1996). In nearly decomposable systems, the effect on overall system performance by component development decisions made cannot be fully established. New information with regard to overall system performance to ensure system stability and performance is required and needs to be incorporated. Clark (1985) argues that there are choices in the development of a design that creates, and is logically prior to, other choices. These antecedents, which may be inherent in the physical structure of the product, create constraints that give rise to further search for alternative designs. They may also arise because of the large number of interdependencies between parts, interdependencies that exert limits on the ability of pre-planning efforts to establish this interconnectedness of system parts. Consequently, establishing all interdependencies, interconnectivity, mutual influence, and aggregated system performance of a complex product platform a priori, is difficult. To manage such complexity, manoeuvrability and flexibility needs to be incorporated into initial plans. Making provision for changes as and when new interdependencies are identified and encountered is important. Learning throughout, the development process is required. This learning is maintained through the process of system aggregation, which also provides feedforward information. Carroll et al. (2002) use the term feedforward to mean predicting future events from a model or theory of action.

System aggregation provides information and knowledge about the effect of development decision made, since overall performance can be established. Aggregation information thus provides a ground for identifying required changes that need to be made to integrated system parts, i.e., components, since they have a direct impact on platform performance. However, it also visualises new opportunities, interdependencies and constraints in relation to the anatomical decomposition of the platform. Changes to the initial anatomical decomposition can be required to incorporate this new knowledge, thereby guiding the following process of component development. The process can be reinitiated until final system version is achieved.

Feedforward output is important in determining new interdependencies and an important contributor to potential decisions to alter already mapped interdependencies. Feedforward output provides increased understanding of what future development paths should look like by visualising both enabling and impeding aggregated systems characteristics. As an enabler, feedforward provides output information and knowledge that brings to light new opportunities for functionality development. Impeding feedforward output is expressed as the limitations to what the system can manage to accommodate in the future. Thus, managing near decomposability in complex platform

development requires a process in which systemic output in the form of feedback and feedforward information can be reincorporated in the evolution of the platform itself.

5.3 Systemic aggregation

System aggregation provides output, which can be applied to increase both component learning and architectural learning (Henderson and Clark, 1990; Helfat and Raubitschek, 2000). MacCormack et al. (2001, p.138) argue that lower level learning results in knowledge that can be directly applied to a specific context, whereas higher-level learning results in deeper knowledge of the process of problem-solving, especially with reference to analysing new frames of reference. Innovation during product development involves learning more about components per se, or learning more about product architectures (Sanchez and Mahoney, 1996). In a nearly decomposable system, there are requirements for the creation and development of component and architectural knowledge simultaneously, as the system evolves.

Integrative activities for the creation of aggregated system versions provide required system-wide feedback and feedforward on overall system component integration. Impact of changes at component and subsystem level will be visible on the higher complete system level. Feedback to components and sub-systems in a nearly decomposable system are provided by the totality of the system and not only from what happens between components and sub-systems within the same hierarchical level (Simon, 1996). System aggregation marks an important step in systems development and provides flexibility since changes can be made in between each aggregation point, thereby limiting system impact to a certain development stage. System aggregation points become an apex on which following development decision can be made. Such an approach carries multiple benefits for the success of the development of complex systems.

The availability of competing development alternatives in the hierarchy of a product or system implicates for subsequent development decisions but choice at the apex has ramifications throughout the hierarchy (Clark, 1985). The choice of apex impact on other aspects of the domain and dominate all others within the domain. That domination arises from the fact that the choice of a core concept creates a set of given conditions with which other parameters must deal (Clark, 1985). In the development of complex platforms, the use of an apex to determine future development paths, its ramifications for the next apex, needs to be exercised frequently.

By providing feedback and feedforward on overall platform performance arising from initially determined interdependencies, information on which to determine future development steps is provided. Flexibility to change existing, or introduce new, system parts is provided from the understanding derived from the way the platform works after having been integrated, allowing further decoupling or new arrangement of functional elements.

The degree to which development processes are analysable or non-analysable has profound implications for how a company decides to approach the task of system development (Lindkvist et al., 1998). One challenge is the need to understand system development implications both at lower component level and at higher system levels. Lower-level quality insurance itself does not prelude higher-level quality insurance. Rather lower-level quality test can only be undertaken to a certain part, leaving aggregated system influence uncared for. Overall system performance can only be

established after system aggregation, which furthermore allows complete system quality tests to be carried out.

Frequent aggregation also carries effects for a company's ability to manage sequential and parallel platform development. Sequential and parallel development can be seen as two opposite approaches to product development. Where and when it is possible to establish interdependencies and their effects, a sequential approach to development should be a straightforward endeavour. Problems sequentially arise when dealing with a complex system, where developers are faced with difficulties in establishing and understanding the effect of all interdependencies and where alteration of existing and emergence of new interdependencies is quite likely to occur. Provisions for instances where system component development needs to be more aligned is required and the ability to simultaneously manage sequential and parallel development becomes important. However, through a development process, which includes several integration and aggregation points, a combined approach can more easily be successfully undertaken.

Further to this, the development of a complex, nearly decomposable system requires the ability to manage both modular and integral system architectures. Modular approaches to development inherently include choice and trade-off between flexibility and rapid innovation, and overall system performance (Ulrich and Eppinger, 1995). However, this proposed trade-off can be questioned on two fronts. For one, a modular approach to develop can indeed have negative consequences for flexibility and rapid innovation. Flexibility impediments results from the limited problem-solving area and because a higher-level system overview is often lacking. Paced development is impeded by the resource-demanding and time-demanding task of integrating a large number of components at one particular point of time in the life of the project. Further to this, the optimisation problem impedes on overall platform quality and performance. Thus, the requirement for simultaneous management of both modular and integral architecture becomes an optimisation issue. When attempting to decompose an integral architecture to a modular one, breaking the complex system into independent and isolated units or modules that can be mixed and matched as selected, sub-optimisation becomes a very vivid threat (e.g., Baldwin and Clark, 2000, 2001; Sanchez and Mahoney, 1996). System aggregation opens up possibilities to make alterations to initial determined functionality add-ons. By being able to establish aggregated platform performance at several points in the development stage, it is also possible to take this input as basis for alterations to future development paths. If function optimisation cannot be established, then perhaps system optimisation should be the focus of attention. Sub-optimisation on functionality level can still be managed to lead to greater aggregated system-wide optimisation.

In addition, we extend the proposition that modularity makes complexity manageable by making it possible to run experiments at the level of modules, rather than the entire artefact, and in parallel (Baldwin and Clark, 2001) into the proposition that it is local independence and aggregated responsiveness, and a combined modular and integrative process, which makes complexity manageable. Whereas 'technological' experimentation indeed takes place at the level of modules, the avenue for this experimentation arises due to the integral architecture evident through the aggregation process.

6 Conclusions

In this paper, we have argued that managing complex platform development often is a matter of managing near decomposability. We have also discussed how ambiguity of interconnectedness and the non-simple interaction between system parts can be managed. Managing complex platform development projects is more than merely a planning exercise. The a priori establishment of a large number of components and the many interrelationships that exist between them are often reduced to a choice of not optimal but good enough choices. Planning goes a long way in ensuring lower-level component quality, but is insufficient in providing an accurate estimation of overall system performance. Lower level quality is not an all inclusive determinant for satisfactory overall system performance.

In an attempt to broaden the managerial and process implications of learning throughout the development process, we also propose that aggregated information, the knowledge it carries, and its implications for learning comes in two shapes. The first is feedback on components, i.e., the feedback on the quality of integrated system parts. The second type is feedforward, information on platform performance. Feedback information is important to compare with, or contrast to, planned activities, plans, mental maps, existing knowledge and to evaluate and determine success. Feedforward is important to establish future development paths, identify new interdependencies, and is a control mechanism that furthermore indicates elusive actions or alterations that are required for future action.

We have shown that one way of tackling the issue of near optimisation is through frequent system aggregation. Iterative functionality testing and verification within and between different systems levels can lead to improved robustness, quality improvements, paced development, and flexibility.

In the development of a nearly decomposable system, the simultaneous need for component independence and aggregated system responsiveness is evident. A platform development approach that can cater to overall system performance regularly throughout the development process is required. The simultaneous provision of stability and flexibility becomes important. Stability requirements are a result of the need to facilitate the decentralisation of development activities and to establish system-wide quality criteria. Flexibility requirements materialise as a response to the need to incorporate new information. The existence of both analysable and non-analysable interdependencies renders strong requirements for the ability to make platform development alterations without impeding on all development work carried out.

Acknowledgements

Previous drafts of this paper were presented at the R&D Management Conference, the CINet Conference, and the IRNOP conference, and at EPOK and KITE seminars at Linköping University, all in 2007. We thank reviewers and audiences at these meeting for their valuable input to this paper. Special thanks are also due to Hans Andersson and Thomas Magnusson for their comments on a draft manuscript. Financial support from Handelsbanken's research foundations and the Bank of Sweden Tercentenary Foundation is gratefully acknowledged.

References

- Baldwin, C.Y. and Clark, K.B. (2000) *Design Rules: The Power Of Modularity*, MIT Press, Cambridge, MA.
- Baldwin, C.Y. and Clark, K.B. (2001) *The Value and Cost of Modularity*, Harvard Business School, Mimeo.
- Brusoni, S., Marengo, L., Prencipe, A. and Valente, M. (2004) *The Value and Costs of Modularity: A Cognitive Perspective*, Brighton: SPRU, 2004, p.22 (SPRU Electronic Working Papers Series; No. 123).
- Carroll, J.S., Rudolph, J.W. and Hatakenaka, S. (2002) *Organizational Learning from Experience in High-Hazard Industries: Problem Investigations As Off-Line Reflective Practice* (March), MIT Sloan Working Paper No. 4359-02, Available at SSRN: <http://ssrn.com/abstract=305718> or DOI: 10.2139/ssrn.305718).
- Clark, K.B. (1985) 'The interaction of design hierarchies and market concepts in technological evolution', *Research Policy*, Vol. 14, pp.235–251.
- Dawidson, O., Karlsson, M. and Trygg, L. (2004) 'Complexity perception – model development and analysis of two technical platform projects in the mobile phone industry', *International Journal of Information Technology and Decision Making*, Vol. 3, No. 3, pp.493–512.
- Dubois, A. and Gadde, L-E. (2002) 'Systemic combining: an abductive approach to case research', *Journal of Business Research*, Vol. 55, pp.553–560.
- Ericsson, A. and Erixon, G. (1999) *Controlling Design Variants*, Society of Manufacturing Engineers, USA.
- Ethiraj, S.K. and Levinthal, D. (2004) 'Modularity and innovation in complex systems', *Management Science*, Vol. 50, No. 2, pp.159–173.
- Frenken, K., Marengo, L. and Valente, M. (1999) 'Interdependencies, near-decomposability and adoption', in Brenner, T. (Ed.): *Computational Techniques for Modelling Learning in Economics*, Kluwer, Boston, pp.145–165.
- Gavetti, G. and Levinthal, D. (2000) 'Looking forward and looking backward: cognitive and experiential search', *Administrative Science Research*, Vol. 45. No. 1, pp.113–137.
- Gawer, A. and Cusumano, M.A. (2002) *Platform Leadership: How Intel, Microsoft and Cisco Drive Industry Innovation*, Harvard Business School Press, Boston MA.
- Halman, J.I.M., Hofer, A.P. and van Vuuren, W. (2003) 'Platform-driven development of product families: linking theory with practice', *Journal of Product Innovation Management*, Vol. 20, pp.149–162.
- Helfat, C.E. and Raubitschek, R.S. (2000) 'Product sequencing: co-evolution of knowledge, capabilities and products', *Strategic Management Journal*, Vol. 21, pp.961–979.
- Henderson, R.M. and Clark, K.B. (1990) 'Architectural innovation: the reconfiguration of existing product technologies and the failure of established firms', *Administrative Science Quarterly*, Vol. 35, pp.9–30.
- Hobday, M. (1998) 'Product complexity, innovation and industrial organisation', *Research Policy*, Vol. 26, pp.689–710.
- Krishnan, V. and Gupta, S. (2001) 'Appropriateness and impact of platform-based product development', *Management Science*, Vol. 47, No. 1, January, pp.52–68.
- Levitt, B. and March, J.G. (1998) 'Organizational learning', *Annual Review of Sociology*, Vol. 14, pp.319–340.
- Lindkvist, L., Söderlund, J. and Tell, F. (1998) 'Managing product development projects: on the significance of fountains and deadlines', *Organization Studies*, Vol. 19, pp.931–951.
- MacCormack, A., Verganti, R. and Iansiti, M. (2001) 'Developing products on internet time: the anatomy of a flexible development process', *Management Science*, Vol. 47, No. 1, pp.133–150.

- Marengo, L., Pasquali, C. and Valente, M. (2005), 'Decomposability and modularity of economic interactions', Published in: Callebaut, W. and Rasskin-Gutman, D. (Eds.): *Modularity: Understanding the Development and Evolution of Complex Natural Systems*, MIT Press, Cambridge MA, pp.835–897.
- Meyer, M.H. (1997) 'Revitalize your product lines through continuous platform renewal', *Research Technology Management*, Vol. 40, No. 2, p.12, 17.
- Meyer, M.H. and DeTore, A. (2001) 'Perspective: creating a platform-based approach for developing new services', *The Journal of Product Innovation Management*, Vol. 18, pp.188–204.
- Meyer, M.H. and Lehnerd, A. (1997) *The Power Of Product Platforms: Building Value And Cost Leadership*, The Free Press, New York, NY.
- Muffatto, R. and Roveda, M. (2000) 'Developing product platforms: analysis of the development process', *Technovation*, Vol. 20, pp.617–630.
- Nelson, S.A., Parkinson, M.B. and Papalambros, P.Y. (2001) 'Multicriteria optimization in product platform design', *Journal of Mechanical Design*, Vol. 123, June, pp.199–204.
- Robertson, D. and Ulrich, K. (1998) 'Planning for product platforms', *Sloan Management Review*, Vol. 39, No. 4, pp.19–31.
- Sanchez, R. and Mahoney, J.T. (1996) 'Modularity, flexibility, and knowledge management in product and organization design', *Strategic Management Journal*, Vol. 17, Special Issue: *Knowledge and the Firm*, pp.63–76.
- Sawhney, M.S. (1998) 'Leveraged high-variety strategies: From portfolio thinking to platform thinking', *Journal of the Academy of Marketing Science*, Vol. 26, No. 1, pp.54–61.
- Shibata, T. and Yano, M. (2002) *Dynamic Evolution of Product Architecture – Alternative View of Technical Progress of Numerical Controller*, National Institute of Informatics, Technical Report (August).
- Simon, H. (1962) 'The architecture of complexity', *Proceedings of the American Philosophical Society*, Vol. 106, No. 6, pp.467–482.
- Simon, H. (1996) *The Sciences of the Artificial*, 3rd ed., MIT Press, Boston MA.
- Simon, H. (1999) 'Can there be a science of complex systems?', in Bar-Yam, Y. (Ed.): *Unifying Themes in Complex Systems: Proceedings from the International Conference on Complex Systems 1997*, Perseus Press, Cambridge MA, pp.4–14.
- Simon, H. (2002) 'Near decomposability and the speed of evolution', *Industrial and Corporate Change*, Vol. 11, No. 3, pp.587–599.
- Sun Microsystems (2004) *Why Software Architecture is Critical for Product Lifecycle Management Success*, White Paper, Accessed 20070216, http://www.ptc.com/WCMS/files/36847/en/36847en_file1.pdf.
- Tatikonda, V.M. (1999) 'An empirical study of platform and derivative product development projects', *Journal of Product Innovation Management*, Vol. 16, pp.3–26.
- Ulrich, K.T. (1995) 'The role of product architecture in the manufacturing firm', *Research Policy*, Vol. 24, pp.419–440.
- Ulrich, K.T. and Eppinger, S.D. (1995) *Product Design and Development*, McGraw-Hill, New York.
- Walsham, G. (1995) 'Interpretive case studies in IS research, nature and method', *European Journal of Information Systems*, Vol. 6, pp.74–81.
- Walsham, G. (2006) 'Doing interpretive research', *European Journal of Information Systems*, Vol. 15, pp.320–330.
- Wheelwright, S.C. and Clark, K.B. (1992) 'Revolutionizing product development', *Quantum Leaps in Speed, Efficiency, and Quality*, The Free Press, New York.
- Yassine, A.A and Wissmann, L.A. (2007) 'The implications of product architecture on the firm', *Systems Engineering*, Vol. 10, No. 2, pp.118–137.