A Systematic Analysis of the Effect of Task Clarity on Software Development Design

Werner Mellis

University of Cologne

Department of information systems

Pohligstr. 1, 50969 Köln, Germany

Tel. +49 221 470-5368

Email: mellis@informatik.uni-koeln.de

ABSTRACT

Two different types of development tasks are distinguished: Clear and unclear development tasks. Based on hypotheses from organizational theory two different designs of software development are derived. The transformational design is appropriate if the development task is clear. In case of an unclear development task software development should employ the adaptive design.

The transformational design conforms to the explicit recommendations and implicit assumptions of process oriented software quality management, a software management style considered by many authors to be the universally valid paradigm of software development. Because of the fundamental differences between the two designs we conclude, that process oriented software quality management is not universally valid and should not be applied to unclear software development tasks.

Keywords
Organizational design, software management, quality management, product development

1 THE CONTINGENCY APPROACH TO SOFTWARE DEVELOPMENT DESIGN

During the last decade process oriented software quality management (PSQM) as represented by standards like Capability Maturity Model (CMM), Bootstrap, ISO 9000, SPICE (ISO 15504) etc. became the celebrated paradigm of software production [3]. There is an almost perfect agreement among experts that for a software producer's success the design of its software processes according to PSQM has to be given highest priority. This conviction is usually formulated as a universal recommendation which is given independent of the type of software developed, the type of customer, the type of application or the branch of industry the organization belongs to.

Nevertheless some authors formulated a sharp criticism of PSQM as even being a serious risk to a company's competitive potential. Case studies of some of the world's most successful software producers demonstrate, that they do not follow the recommendations of PSQM.

Analyzing the empirical evidence for PSQM and its criticism demonstrate, that there is empirical evidence for the relevance of PSQM as well as for the software development design found in case studies like [2] or [4]. Therefore it is concluded, that neither PSQM nor the software development design found in those case studies can be considered universal, but are appropriate in certain situations characterized by some contingency factors. For a more detailed discussion see [7].

In order to flesh out this contingency approach, three questions need to be answered.

1. Which contingency factors are effecting software development design?

2. What are the essential characteristics of the different software development designs?

3. Which hypotheses are relevant to justify or explain the different software development designs?

In this paper three partial answers are given to the three questions.

1. A contingency factor (task clarity) is described and it is argued, that clear and unclear development tasks demand different software development designs, called transformational and adaptive design of software development respectively.

2. The transformational and adaptive design of software development are described by well known dimensions of organizational design. The transformational design conforms to the explicit recommendations and implicit assumptions of PSQM. The adaptive design is in accordance to the software development design found in case studies e.g. [2] and [4].

3. The derivation of the two different designs of software development is based on the well established hypotheses of organizational design.

2 CLEAR AND UNCLEAR DEVELOPMENT TASKS

There are several different factors reducing the clarity of a development task [10].

1. Unclearness of Requirements

Customer and technological requirements are unclear.

2. Dynamic Nature of Requirements

Customer and technological requirements change during development time.

3. Technology Dynamics

Knowledge about software technology and software process technology change during development time.

A development task is called unclear, if it is affected significantly by these three factors. Else it is called clear.

In the following requirement refers to any request by a customer, a partner, or the technical environment, that some attribute of the software to be developed has a specific value.

A requirement is clear, if an attribute and its possible values are known (to the development personnel) and it can be determined with acceptable effort independent of development activities (analyzing design alternatives, reworking the design, prototyping, implementation trials etc.), which of the possible values ought to be realized. If the development personnel have decided, which value ought to be realized, the requirement is called known.

There are two different cases of unclear requirements.

1. A requirement is unclear, if some attribute of the software to be developed and its possible values are known, but with acceptable effort and without development activities it cannot be decided, which value should be advantageous at release time, because the consequences of the different values are unknown or there are no criteria to estimate these consequences.

2. In the other case only from development experience or from customer feedback it is noticed that some attribute is relevant.

If a requirement is clear but unknown, then it is possible without development activities to make it known with an acceptable effort. If a requirement is not clear, then without experience from the development activities or without application of development results the requirement can only be determined with unacceptable effort.

Remark: 1. It should be clear, that in practical settings tasks are always more or less clear. However, in order to ease understanding the argument, here an idealized binary distinction is used. 2. There is a well known categorization of programs into three categories: S-, P- and E-programs depending on how they are related to reality. Here we are concerned with E-programs only, which “mechanize a human or societal activity” [5]. Lehman suggests, “that it is always possible to continue the system partitioning process until all modules are implementable as S-programs” [5]. In [6] Lehman states an hypothesis under investigation, explaining the limited success of software process improvement projects to realize this suggestion. Here we point out a condition under which it is practically infeasible to follow this suggestion and derive recommendations for the organization of software development under this condition.

3 SYSTEMATIC DERIVATION OF SOFTWARE DEVELOPMENT DESIGN RECOMMENDATIONS

There is a well established organizational theory, which has been applied successfully in many other cases. See for example [9]. In order to arrive at a justified proposal, we apply this body of organizational knowledge to the problem of software development design in case of clear and unclear development task.

Design Dimensions and Design Procedure

For the development of a suggestion on software development design with clear respectively unclear development tasks among others the following design dimensions must be considered:

- Process organization,
- Division of labor and organizational structure,
- Coordination,
- Planning and control,
- Leadership and motivation,
- Communication,
- Quality assurance,
- Communication interface to the customer.

For some of the dimensions listed above a substantial alternative is derived. Based on well established organizational hypotheses we argue for a recommendation of software development design for clear and unclear development tasks.

The application of the organizational hypotheses is based on a task analysis. This provides us with information about the various subtasks of software development, possible forms of the division of labor, interdependencies between the subtasks and the obstacles to a reliable, complete and efficient supply of information between them. This information is necessary to decide between the alternatives of process design and division of labor.

The task of software development can be broken down into subtasks according to the activity or according to the structure of the product. If both types of task break down are combined the smallest tasks are the application of some activity to some module, e.g. analysis of some feature, design of some component, implementation of some component. Further there are some tasks, which are not related to individual components like e.g. configuration manage-

ment, project staffing, architecture design or component integration.

In the following these tasks are called elementary subtasks (see fig. 1) though they could further be broken down for example according to work phases like planning and execution.

Information flows

Because of the complexity and immaterial nature of software products the subtasks' outputs are extensive amounts of information, which need to be supplied as input for other subtasks. Most of this information is transferred in coded form, because it needs to be exchanged between people or cannot be directly processed by humans. If it is transferred in coded form it must be coded, transported and decoded. The transportation is without any problem. But coding (formulation) and decoding (understanding) can demand huge effort and can cause significant risk of error, leading to defects in a subtask's input (lacking, false or irrelevant information).

This is especially true for the software specification (the result of the analysis) in case of an unclear development task. Because of the amount and unclearness of the requirements and because of the dynamics of the requirements and the technology, the specification is incomplete and unstable.

Good back-ground knowledge about the application and the technology can partly compensate for the specification's incompleteness. But this knowledge is learnt from experience over a long time and must be comprehended as "tacit" in the sense of Nonaka and Takeuchi. Typically it is acquired by a person who intensively deals copes with the application or the technology. It is "sticky" i.e. separating it from one person and transferring it to another person demands substantial effort.

This has to be taken into account in designing the division of labor. If in case of an unclear development task analysis on the one hand and design and implementation on the other hand is not the same person's responsibility, then we have to face a very high effort to avoid incompleteness of the specification or to transfer the back-ground knowledge about the application or we have to face significant risk of defects.

Interdependencies

Let us first distinguish horizontal interdependencies between the elementary subtasks on the same component, like analysis, design, implementation and test of one component, and vertical interdependencies between the elementary subtasks with the same activity, like implementation of component a and implementation of component b.

Usually it is assumed, that there is a sequential horizontal interdependency. I.e. results of one subtask are the basis of the following. This sequential horizontal interdependency among the subtasks broken down according to the activity exists only under certain circumstances. The independence of e.g. the analysis from the following subtasks presupposes, that every requirement is clear and that the technology is well known.

In case of an unclear development task, there are reciprocal interdependencies between analysis and design. I.e. results of the subtask analysis is a necessary input for the subtask design and results of the subtask design is a necessary input for the subtask analysis [WaEC93]. The reason for the reciprocal horizontal interdependencies is, that intensive analysis of design alternatives is essential for the clarification of requirements. "The discussion of requirements from this point, however, was rooted in the context of specific design alternatives (‚Plan A vs. Plan W')".[12] Analysis of design alternatives shows their different sets of features allowing to learn about relevant requirements and limiting the search for requirements to those necessary to discriminate between the design alternatives. Thus in case of unclear requirements analysis is not independent from design.

Further reciprocal horizontal interdependencies are rooted in high technology dynamics. in order to be able to decide design and implementation alternatives knowledge about the technologies employed is necessary. If it lacks because of the innovative nature of the employed technology it needs to be generated by implementation trials (prototypes, simplified versions) or usability test. I.e. results of the implementation subtask respectively. of the test subtask is needed as basis for analysis and design. Therefore there is a reciprocal horizontal interdependency also between analysis and design on the one side and test and implementation on the other.

In case of a clear development task the various requirements are mapped on the modules of a hierarchical architecture. Since requirements and technology are stable, there are no vertical interdependencies between the elementary subtasks after architectural design.

In case of unclear development task there are reciprocal vertical interdependencies between e.g. implementation of component a and implementation of component b. If for example an unclear requirement turns out to refer to several components, this may lead to changing several components, where one component's change depends on the others and vice versa. Similar reciprocal vertical interdependencies may origin in the change of requirements and the growing experience with the employed software technologies.

Fig. 1 is a simplified map of the information flows accompanied by the various interdependencies between the elementary subtasks.

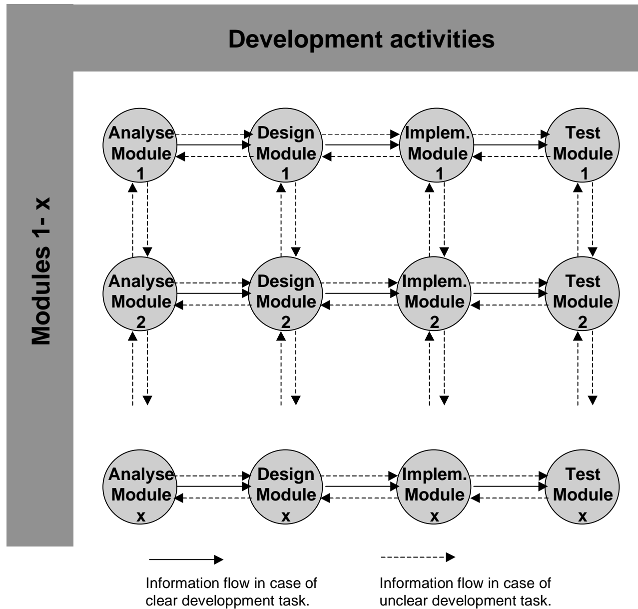

Figure 1: Subtasks and information flows

4 SOFTWARE DEVELOPMENT DESIGN RECOMMENDATIONS

Design recommendations can be given for any dimension of software development design (for an extensive discussion of development in case of unclear development task see [8]). Here we are restricted to some design dimensions essential to allow a clear cut contrast between the two design recommendations in case of clear and unclear development.

Process Design

In process design elementary subtasks are combined to process steps. In software development we build process steps by combining those subtasks, where the same activity is being performed on different components. The resulting process steps are analysis, design, implementation, and test.

Depending upon the temporal arrangement of the process steps, there are two different forms of process organization, the sequential and the parallel process organization.

Beside the temporal arrangement of the process steps the process organization can be distinguished according to the number of increments. A process organization is called incremental, if the product is developed in a sequence of versions or increments, which differ in that each version fulfills an increasing subclass of requirements. If there is only one increment, the process organization is called batch.

In a sequential process organization, the different process steps are executed sequentially without overlap in time. Software integration, i.e. the combination of the different components of the product to the executable total product, is generally carried out at a fixed time after the completion of the development activities.

A process organization is sequential and incremental, if the product is developed in several increments, which are developed sequentially each.

When employing a parallel process organization, the execution of the process steps analysis, design, implementation, and test is overlapping in time. Software integration is done in parallel to those steps and is incremental. Early during development, modules sufficient for a core system with reduced functionality are developed and integrated into an executable basic system. The further development of existing modules occurs in small steps. If some requirements become known, as frequently as possible design, implementation and the integrated system are updated and tested. [2]. Therefore a parallel process organization is always incremental.

Increments in a parallel and in a sequential process organization differ substantially. In a sequential and incremental process organization there are few, substantial increments, whose development can be considered as separately planned projects, representing defined stages of functionality. In a parallel and incremental process organization, there is a vast amount of small increments, representing the daily progress of the development work.

In case of a clear development task a sequential process organization is recommended, while in case of an unclear development tasks a parallel process organization should be preferred.

Usually a sequential process organization is considered advantageous in respect to productivity, product quality and reliability of development planning. This is based on the assumption, that the tasks of any process step can be planned in detail and solved completely within time and budget, if only at the beginning of the process step the necessary input information is detailed, complete and reliable. In case of a clear development task the process step analysis can deliver a detailed, complete and reliable result and therefore – applying the assumption recursively – all following steps can do so. Therefore, if the requirements do not undergo change during development time, the perfect satisfaction of requirements by the product, the avoidance of rework and interrupt and the prevention of uncertainties in planning is just a matter of the careful execution of the various development activities.

In case of unclear development task on the other hand there are reciprocal interdependencies between the development tasks of analysis, design, implementation, and test. Since none of two reciprocally interdependent subtasks can be completed before the other begins, they must be executed overlapping in time.

The higher the dynamics of requirements and technology the more important is the amount of overlap. Let us introduce the “concept freeze” to indicate the point in time, from which on the definition of the product is no longer changed, and the “response time” to indicate the period of

time from concept freeze up to the delivery of the product [4]. Than the response time represents the period of time during which the product cannot be adapted to the changing requirements and technology. In case of a high dynamics of requirements and technology the product will be outdated to some extend, indicated by the response time: the shorter the response time, the less outdated the product. Since in a sequential process organization the response time extends from the beginning of design to the product's delivery, a sequential process organization leads to a relatively outdated product.

A parallel process organization with parallel software integration provides a first product version early during development, which afterwards develops gradually to the final version. This allows a result oriented control of the development progress and a continuous feedback from customers. While continuous feedback is important in case of an unclear development task, result oriented control is also of interest in case of a clear development task.

Design of Positions and Organizational Structure

There are two different approaches to combine elementary subtasks into higher-level tasks and to assign them to organizational units (positions, groups, departments etc.), which result in two distinct forms of division of labor [9].

In an activity oriented division of labor elementary subtasks with the same activity are combined to the higher-level tasks analysis, design, implementation, and test, which are assigned to different organizational units on the same level of hierarchy. In this case there are units specialized on the tasks analysis, design, implementation, or test.

In a product oriented division of labor elementary subtasks with the same component are combined to higher-level tasks like development of component a or development of component b. If those higher-level tasks are assigned to different organizational units on the same level of hierarchy, than there are units specialized on the development of different components.

In case of a clear development task the activity oriented division of labor is recommended. This has two main reasons. Firstly, because of the sequential interdependency the tasks analysis, design, implementation and test can be divided among different organizational units without causing coordination need. Secondly, the different units can specialize in the different activities, which therefore can be performed faster, more efficient and with less risk of failure.

In the case of unclear development tasks, a product-oriented division of labor is recommended, i.e. the complete task of developing a piece of software be divided and assigned to organizational units in accordance with the modular structure of the product's architecture. While a single organizational unit develops each module, it is possible for one organizational unit to be responsible for the development of several architectural modules. This approach is called modular product development [11].

Analysis of the information flows and interdependencies between the elementary tasks in cases of unclear development tasks has shown that the transfer of information between the activities of analysis, design, implementation, and testing is costly and prone to error.

On the other hand, the necessity for the bilateral transfer of information between the elementary tasks within the context of a certain activity can be kept to a minimum through the use of a modular product architecture.

Usually, it is assumed that there is a more intensive exchange of information within organizational units than between different units [9]. For this reason, during unclear development tasks, the product-oriented division of labor is better suited to dealing with the required exchange of information between elementary tasks and more effective in reducing the risk involved with this exchange.

The increase of development costs as a result of the frequency of requirements changes is higher in the case of a process-oriented division of labor than in the case of a product-oriented one, because the costs of every single change are higher. This results from the fact, that, given a product-oriented division of labor, ideally only one organizational unit is involved in the change, as opposed to all or several as in a process-oriented one.

Quality Assurance

We distinguish four dimensions of the design of quality assurance: type, centralization, formalization and timing.

There are two different types of quality assurance: verification, i.e. checking whether the product conforms to explicitly given specifications, or validation, i.e. checking whether the product is useful, when applied under realistic application conditions.

Centralization refers to the extend to which decision making power is distributed. [9] Quality assurance is called centralized, if one person has all the power of decision making concerning quality assurance. It is called distributed, if part of the decision making power, for example decisions about the quality of individual modules of the product, is delegated e.g. to their developers.

Quality assurance is called formal, if the procedures of quality assurance are regulated by detailed rules. It is called informal, if there is only little regulation by rules.

Finally quality assurance can be distinguished according to its timing. The timing is final, if quality assurance is organized as a final phase of a process step, the output quality of which it has to assure. It is called integrated, if it takes place concurrently during the process.

In case of a clear development task a verifying, centralized, formalized, final quality assurance is recommended. The

verifying, final design of quality assurance is well known under the name V-model of testing coined by Boehm. The advantage of the V-model of testing in case of a clear development task is, that it allows to certify, that the product conforms to the specification. It further is very efficient, in that it identifies defects as early as possible and searches different types of defects only once in its various steps. Since product quality in case of a clear development task means conformance to specification, it is important to be able to carefully control the application of quality assurance. In order to achieve this, the procedures of quality assurance are formalized in detailed. The reason to centralize the decision making power lies in the fact, that centralization is the tightest means of coordination of decision making, which therefore supports the controlled and consistent application of quality assurance.

In case of an unclear development task a validating, decentralized, informal, integrated quality assurance is recommended. In case of an unclear development task a verifying quality assurance is possible only in case the specification is written after the product is developed. Since this would increase response time substantially, a verifying quality assurance is impossible. Further the validating, integrated quality assurance, as for example in form of usability tests, has the advantage, that it helps to identify and understand requirements at a time, when it still can help to guide development. The decentralized and integrated quality assurance means, that a developer, responsible for some feature, receives direct feedback about his work and the value it has for a customer. This is usually seen as a stimulus to motivation. [2] Further in case of an unclear development task, the feedback given by an early validation of the work is essential in order to identify or clarify requirements. It is assumed, that the person, who developed the validated version or prototype, has the best knowledge of admissible alternative designs of his feature. If the developer and the quality assurance person are different people, then there is further the risk of defects in communicating the feedback. In case of an unclear development task the quality of quality assurance depends more on the creative construction of test scenarios than on a careful execution of a prescribed procedure. Since formalization may have a negative effect on creativity, it is not recommended.

Interface to the Customer

We distinguish three dimensions of the design of the analysis, which are not already treated above: centralization, formalization and communication. Analysis is called centralized, if one person has all the power of decision making concerning requirements. It is called distributed, if part of the decision making power, concerning for example decisions about the detailed requirements on most features of the product, is delegated e.g. to their developers. Analysis is called formal, if the procedures of analysis are regulated by detailed rules. It is called informal, if there is only little regulation by rules. The dimension of communication is structured in accordance with Clark and Fujimoto's classification of the intra-organizational communication relations [1] by means of three dimensions (the other dimensions are irrelevant or covered elsewhere):

- Frequency of information transmission: A single requirement is considered once vs. is dealt with frequently.
- Direction of communication: unidirectional, only providing information vs. bi-directional, exchanging information.
- Richness of information media: written, impersonal vs. verbal, personal.

In case of a clear development task a centralized and formalized analysis is recommended, where the communication is intended as a unidirectional, written, complete information transmission touching on any requirement only once.

A centralized and formalized analysis supports to achieve a complete, reliable and consistent specification, which is of prior importance, if a clear development task is given. If requirements change, what cannot be prevented completely, such organization helps to maintain requirements' consistency, which is essential. Since the requirements are clear, they can be written down formally. The information transmission can be unidirectional, because the requirements can be derived from a well understood application context. No specific input about the possible technical solutions is necessary. Unless a requirement changes during development, it needs only be stated once.

In case of an unclear development task a decentralized and informal analysis is recommended. The information transmission should be bidirectional, much of it in form of a presentation of a version or prototype and feedback by the customer. Feedback should not be given by formal, written statements, but by observing and discussing the use a customer makes of a version or prototype. Thus communication is rich and informal. The same requirements may be dealt with frequently in order to communicate partial or preliminary information.

In case of an unclear development task requirements analysis to a great extend depends on the analysis of design alternatives and implementation experience. The developer's creativity and their ability to receive and interpret feedback and his ability to realize obscured hints concerning the application conditions is essential for the quality of requirements engineering. Therefore requirements engineering is informal. The same requirement may be dealt with frequently, since several steps may be necessary to clarify it or since it may change.

In order to receive feedback developers must confront the users with early versions or prototypes, thus information transmission is bidirectional. Feedback may be received as formal written statement. But since it can not be assumed

that the user is able to express his requirements easily, other forms of receiving feedback, like for example by observing a users behavior in a usability test, are recommended.

In order to allow a fast reaction to the changing requirements and technology analysis is decentralized. The decision making power concerning the requirements on some module is largely delegated to the developer responsible for this module.

Coordination

Usually five different coordination mechanisms are distinguished: standardization of work process, standardization of work product, standardization of qualification, direct supervision and mutual adjustment. The three standardization mechanisms allow to do the coordination before the work is actually done, while the last two are ad hoc coordination mechanisms, used if during the work is actually done coordination turns out to be necessary.

In case of a clear development task it is recommended to do the coordination in advance by employing the three standardization mechanisms. This is generally advantageous, because it improves the reliability of planning. Usually all three standardization mechanisms are employed. Of specific importance in case of software development is standardization of work product by use of an architecture. This is done by mapping the requirements on to the various modules given by the architecture and formally describing the interfaces between the different modules. Further standardization of work process, which is the essence of quality management, is of significant importance, to prevent the need for ad hoc coordination.

In case of an unclear development task coordination in advance by standardization is limited. Standardization of work process is limited, since in case of an unclear development task a much higher degree of variation of the various processes is necessary. Further there are no written requirements, which can be used for coordination early during development.

While in case of a clear development task ideally no ad hoc coordination is necessary, it plays a significant role in case of an unclear development task. In order to reduce the coordination need, the division of labor is based on a modular architecture as described above. But because of the continuous clarification and change of requirements in case of an unclear development task even with a modular architecture many decisions of software design, implementation and test need to be coordinated ad hoc between the organizational units responsible for the development of the different software modules.

To satisfy the need for ad hoc coordination the two mechanisms direct supervision or mutual adjustment can be employed. In case of an unclear development task mutual adjustment is of prior importance.

The employment of direct supervision as a coordination mechanism means that coordinating decisions are made by some decision maker hierarchically above the organizational units whose work needs to be coordinated. Most of the relevant information on which the decision has to be based, will be accumulated in the organizational units below. Much of this information is technically complicated or tacit knowledge e.g. background knowledge acquired over a longer period of time, which cannot easily be completely codified and communicated. Therefore frequently extensive amounts of complicated, sticky information need to be communicated and understood by the decision maker, who will most likely turn out to be a severe bottleneck. For this reason coordination is based on mutual adjustment, which under the conditions of an unclear development task leads to faster decisions of better quality.

5 CONCLUSION

Two different software development designs have been described as opposing designs in respect to some well known dimensions of organizational design.

Therefore naming of the two designs of software development is arbitrary to some extend. They differ in any of the described design dimensions. Therefore the naming could be based on various design dimensions. Stressing the contrariety in process organization they could be named: Sequential and parallel or concurrent development. If it is intended to stress the differences in the division of labor, they could be distinguished as process oriented development versus product oriented or modular development.

Here it is intended to emphasize the different approaches to satisfying requirements. The name transformational design should point to the fact, that the model assumes explicitly stated requirements, which are transformed into a software, with certified conformance to the requirements.

The name adaptive design on the other hand should stress the fact, that early during development according to the adaptive model a version of the product is developed and gradually adapted to the application conditions, without the use of explicitly stated requirements. For this purpose software development according to the adaptive design is organized in a way to support receiving feedback and to quickly improve the fit of the existing version to the real needs. For this reason changes to the software are integrated into the running version as quickly as possible. This allows to base testing, analysis of detailed requirements, but also project controlling on the running version of the product, without being forced to explicitly state requirements in an early closed phase of software development.

The two designs are recommended in case of a clear and an unclear development task respectively. This recommendation is justified on basis of well established hypotheses of organizational design. From the derivation of the two designs of software development it is also evident, that the two different designs cannot replace each other. In case of

an unclear development task, early termination of requirements analysis would not lead to clear, complete and stable requirements, but to badly understood requirements. Therefore, if applied under time pressure, it would produce an unsatisfactory product. On the other hand in case of a clear development task, employing the adaptive model most likely yields inferior predictability in respect to time and costs and also inferior quality in the sense of conformance to explicitly stated requirements.

6 REFERENCES

[1] *Clark, Kim B.; Fujimoto, Takahiro:* Product Development Performance: Strategy, Organization, and Management in the World Auto Industry. Harvard Business School Press, Boston 1991.

[2] *Cusumano, Michael. A.; Selby, Richard W.:* Microsoft Secrets. The Free Press, New York 1995.

[3] *Fox, C.; Frakes, W.:* The Quality Approach: Is It Delivering? in Communications of the ACM, 40 (1997) 6, pp. 25-29.

[4] *Iansiti, Marco; MacCormack, Alan:* Developing Products on Internet Time. In: HARVARD BUSINESS REVIEW 75 (1997) 5, pp. 108-117.

[5] *Lehman, Meir M.*: Programs, Life Cycles, and Law of Software Evolution. Proceedings of the IEEE, Vol. 68, No. 9, 1980, pp. 1060-1076.

[6] *Lehman, Meir M.*: Feedback in the software evolution process. Information and Software Technology 38, 1996, pp. 681-686.

[7] *Mellis, Werner:* Software Quality Management in Turbulent Times - Are there Alternatives to Process oriented Software Quality Management? Accepted for publication in: SOFTWARE QUALITY JOURNAL

[8] *Mellis, W., Herzwurm, G., Müller, U., Schlang,H., Schockert, S., Trittmann, R.:* Concurrent Software Development. Shaker, Aachen 2000

[9] *Mintzberg, Henry:* The Structuring of Organizations. A Synthesis of the Research. Prentice-Hall, Englewood Cliffs/ N.J. 1979.

[10] *Picot, Arnold; Reichwald, Ralf; Nippa, M.:* Zur Bedeutung der Entwicklungsaufgabe für die Entwicklungszeit: Ansätze für die Entwicklungsgestaltung. In: ZEITSCHRIFT FÜR BETRIEBSWIRTSCHAFT-Sonderheft 23: Zeitmanagement in Forschung und Entwicklung (1998), pp. 112-137.

[11] *Sanchez, Ron; Mahoney, Joseph T.:* Modularity, Flexibility and Knowledge Management in Product and Organization Design. In: IEEE ENGINEERING MANAGEMENT REVIEW 25 (1997) 4, pp. 50-61.

[12] *Walz, Diane .B.; Elam, Joyce J.; Curtis, Bill:* Inside a Software Design Team: Knowledge Acquisition, Sharing , and Integration. In: Communications of the ACM 36 (1993) 10, pp. 63-77.