Features and Design Patterns – A Comparison

Florian Arnold, Gerd Podehl

Research Group for Computer Application in Engineering Design (Prof. C.W. Dankwort)
Department of Mechanical and Chemical Engineering, University of Kaiserslautern
Erwin-Schroedinger-Str., D-67653 Kaiserslautern, Germany
{arnold, podehl}@mv.uni-kl.de

Abstract

Today, the worlds and terminologies of mechanical engineering and software engineering coexist, but they do not always work together seamlessly. Both worlds have developed their own separate formal vocabulary for expressing their concepts as well as for capturing and communicating their respective domain knowledge. But, these two vocabularies are not unified, interwoven, or at least interconnected in a reasonable manner. Thus, the subject of this paper is a comparison of the vocabularies of the two fields, namely feature technology from the area of mechanical engineering and software design patterns from the software engineering domain. Therefore, a certain amount of definitions, history, examples, etc. is presented for features as well as for design patterns. After this, an analysis is carried out to identify analogies and differences. The main intention of this paper is to inform both worlds – mechanical and software engineering – about the other side's terminology and to start a discussion about potential mutual benefits and possibilities to bridge the gap between these two worlds, e.g. to improve the manageability of CAx product development processes.

Keywords: Feature Technology, Design Patterns, Mechanical Engineering, Software Engineering.

1 Introduction

Just about every imaginable niche in nature has been filled by creatures which have independently come up with similar biological solutions to the same ecological problems [30]. As the famous naturalist Charles Darwin [16] put it, *"I am inclined to believe that in nearly the same way as two men have sometimes independently hit on the very same invention, so natural selection, working for the good of each being and taking advantage of analogous variations, has sometimes modified in very nearly the same manner two parts in two organic beings, (...)"*. Over millenniums, some creatures arrive independently at the same solution because it is the solution that works most efficiently. None of the matches are exact, but all employ the same principles. Biologists call this duplication of shape and behaviors in unrelated animals *convergent evolution*.

So, keeping the definition of the term *convergent evolution* in mind, let us regard the two worlds of mechanical engineering (ME) and software engineering[1] (SE): Is there a convergent evolution of *feature technology* from the ME area and *software design patterns* from the SE domain? To clarify this and other questions, we are going to examine the analogies and differences in terms, definitions, semantics, etc. of features and design patterns.

One of the essentials that every engineering discipline must possess is a common vocabulary for expressing its concepts and a language to define relationships between the individual words.

The process chains in ME and SE from conceptual design to product maintenance have reached a scale where it is impossible to rely only on personal communication between single individuals. For a modern company which often has to deal with both engineering worlds, a clear and efficient information

[1] IEEE Standard Computer Dictionary (1990): Software engineering is the application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software.

flow within and between these worlds is vital for success.

In any case, the used terms and languages for communication must be

- as *formal* as possible for reasons of precision, consistency, and reproducibility;
- as *simple* as possible for easy learning and handling;
- as *flexible* as possible for allowing abstraction with different levels of detail, and the adoption to different contexts.

Both worlds have developed their own formal languages with meaningful vocabulary in order to reach these goals and to capture domain knowledge. While ME as the older discipline is very much used to working with standard part catalogues, norms and fixed rules for documentation and drawings, SE is a young science where the re-use of products is still an issue. Nevertheless, with the object-oriented approach and other techniques, SE has overcome the low level vocabulary of data structures and flow charts by introducing high level modeling languages and design patterns with arbitrary levels of abstraction.

Today, the worlds and languages of ME and SE coexist but do not really interact. In most cases the involved persons do not even know each other's concepts. This causes problems when the two worlds meet and computer scientists who are not aware of ME needs shall design CAx systems for mechanical engineers who do not care about software development issues.

As being one of the most advanced approaches to support the mechanical engineers in their specific environment, feature technology clearly reveals the need for a sound understanding of the requirements deriving from the complex process of product design.

The goal of this paper is to inform both worlds – mechanical and software engineering – about the other side's language and to search for opportunities where one area can learn from the experiences and developments already made within and by the other. If engineers could find a commonly understandable language for describing their work and then use it for improving feature technology, mechanical engineering could derive great benefit from it. As a basis for comparison now follows an overview of features and design patterns as well as a short description of the two different "languages".

2 Features in Mechanical Engineering

2.1 From Drawings to Features

Before computers entered the domain of mechanical engineering, there already existed a sophisticated science of how to design the shape of a part, how to calculate its dimensions, and how to apply different techniques to finally manufacture it. The engineers sufficiently solved the problem of communicating the design intent throughout the development and manufacturing process chain by using technical drawings. Implementing strict rules for the creation, use and maintenance of those two-dimensional drawings, they have been able to avoid ambiguity for their three-dimensional tasks. Standardized drawings allowed it to store information about parts that can easily be re-used for different applications. The knowledge about catalogues of standard design solutions and a well structured procedure of solving new design problems (like the German VDI2221 and VDI2222) are still essential for a good engineer.

Computers initially entered the scene of ME mainly for performing mathematical calculations followed by controllers for manufacturing machines, and with the abilities of graphical user interfaces they quickly substituted the traditional manual drawings, due to their ability to easily perform fast changes. This main advantage – easy and fast changes – has eventually lead to today's 3D variational design systems and other specialized computer-aided systems for almost all steps within the process chain. Even the process itself is concerned when PDM and EDM systems are used to handle its working procedures and information flows.

The concept of features was introduced in order to provide the CAD user with meaningful entities which are much more related to his engineering background than to the computer graphics environment. Working with engineering primitives [29] like drill holes and wells makes much more sense than defining cylinders; the user designs his model "by features". If those features can be restricted by reasonable constraints they facilitate the work of the designer who does not need to check critical model parameters because the system can automatically handle this for him.

Nevertheless, the use of computers did not necessarily improve the flow of information [14], and the resulting data is far from being non-ambiguous as soon as data is transferred from one

application to the other. This is mainly due to the lack of standards for data which is not directly related to pure geometry (even this area is not fully standardized), but represents aspects of the model that are of higher semantic meaning.

We do not want to go into detail of all the theoretical and applicatory problems of data exchange and the three main directions of feature applications which are *design by features*, *feature recognition*, and *feature mapping*, but demonstrate how a consistent use of feature technology could re-gain some kind of common language for carrying information throughout the process chain [31], [15]. In order to achieve this high goal, features must not be restricted to always dealing with geometric properties of the product model. They are not only more than 3D primitives, so-called *form features*, as some definitions imply (e.g. *feature = form feature & semantics* [20]), but can also consist of pure non-geometric information, like material, cost, or even purpose.

2.2 Feature Definition by FEMEX

With this background, the feature approach given by FEMEX (*FEature Modelling EXperts*)[2] could provide a theoretical frame which does not devaluate recent feature system developments, but also opens up the possibility of using the concept of features in the whole process chain. According to this definition, a feature can be described as follows [33]:

1. A feature is an information element representing a region of interest within a product.
2. A feature is described by an aggregation of properties of a product. The description contains the relevant properties including their values and their relations (i.e. structure and constraints).
3. A feature is defined in the scope of a specific view onto the product description with respect to the classes of properties and to the phases of the product life cycle (Figure 1).

This definition bears three very important implications. First, a feature is no longer a physical region of a part, but only existent in the world of information technology as the representation of some important aspects of the part. Second, geometry is only one class of properties among others. And third, features are view-dependent within the process, which itself becomes an issue through this definition.

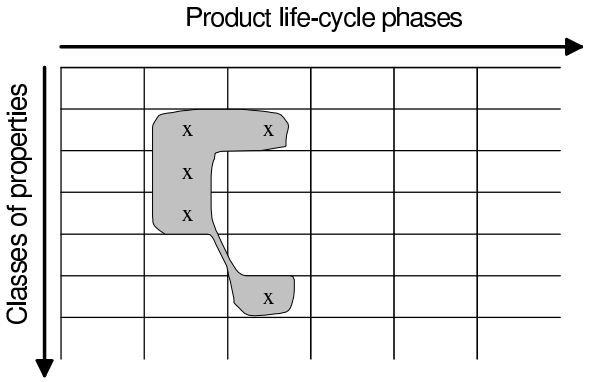

Figure 1: A view defining a feature within the matrix representation of property classes and product life cycle phases [33]

3 Design Patterns

3.1 A Brief History of Patterns

In the short but rapidly evolving history of computer sciences and software engineering the used techniques, methods, processes, means and facilities have changed very much. Projects often failed because developers were unable to communicate good software designs, architectures, and programming practices to each other. Not many years ago, data structures, flow charts and modular programming techniques ruled the scene. Then, the object-oriented paradigm started its triumphal procession. In the context of object-oriented software development, design patterns for software development became one of the "hottest" topics in the software engineering area in the past years.

Patterns have roots in many disciplines, most notably in the writings of the architect Christopher Alexander who has written several books on the topic as it relates to urban planning and building architecture [2], [3]: *"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million*

[2] FEMEX is a group of several international research institutes, system suppliers and application companies with the aim of harmonizing the international feature activities [5], [9], [26], [25].

times over, without ever doing it the same way twice". But a pattern does more than just identify a solution, it also explains why the solution is needed! Therefore, patterns serve as abstractions which contain domain knowledge and experience and which systematically document abstractions.

Software patterns first became popular with the object-oriented "Design Patterns" book by the "Gang-of-Four" (GoF) [18]. These patterns by the GoF are sometimes called *non-generative* because they were observed in systems that had already been built.

The process of looking for patterns to document is called *pattern mining* (or sometimes *reverse-architecting*). Patterns that generate systems or parts of systems, i.e. patterns that people can use when building systems, are called *generative patterns* [12].

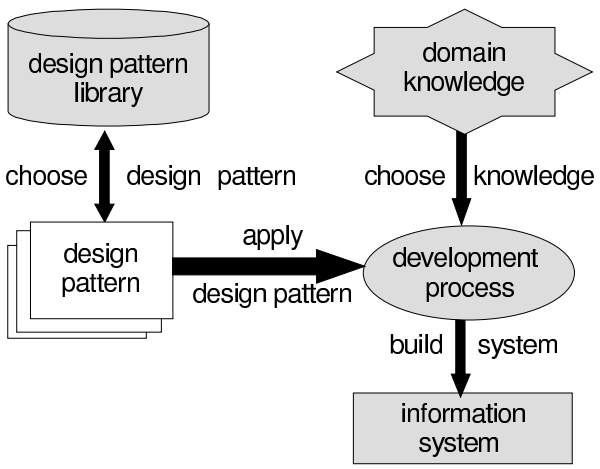

Figure 2: Applying design patterns during the software development process [7].

Meanwhile, the large amount of existing design patterns covers disciplines and domains like distributed [24] and concurrent systems, reliability, real-time systems, business and electronic commerce, organizational design, software reuse [17], and interaction design (including user-interface design [27]).

In the software engineering domain, patterns can also be applied to [21]: Programming idioms, coding idioms, data structures, algorithms, protocols, development of new frameworks (sets of extensible classes), use of existing frameworks, analysis models, system architecture, reconstruction of software, development organization and development process.

A relatively new phenomenon are "bad" patterns that represent a "lesson learned": *Anti-patterns* [8] are targeting common errors and issues in software development and project management that can cause a project to fail and try to give practical guidelines on refactoring solutions that correct them. They are based on the idea that it is often just as important to see and understand bad solutions as it is to see and understand good ones.

Many pattern description formats use graphs or graphical notations, e.g. the Unified Modeling Language (UML) [6], to express their structure and dynamic behavior using visual aids such as class, sequence, collaboration, and use case diagrams.

The general practice of applying design patterns during the software development process is shown in Figure 2.

3.2 A Definition of Patterns

According to [21], a pattern can be defined as *"a solution to a problem in a context"* (see Figure 3) whereby

- ***Context*** refers to a recurring set of situations in which the pattern applies. Thereby all relevant context parameters have to be taken into account;
- ***Problem*** refers to a set of ***forces*** – goals and constraints – that occur in that context;
- ***Solution*** refers to a ***canonical design form*** or ***design rule*** that someone can apply to resolve these forces.

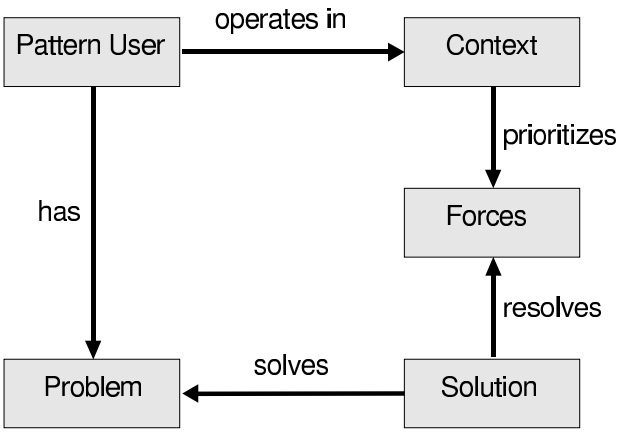

Figure 3: A pattern definition [23]

A pattern in the so-called *canonical form* [10] should additionally comprise the following essential

components (either directly or recognizable upon reading a pattern):

- A name,
- a few examples,
- the resulting context: What forces it leaves unresolved,
- the design rationale: Where the pattern came from,
- why it works,
- why experts use it,
- related patterns, and
- known uses.

Patterns are not a software development method or process. Nevertheless, during the software development process, they complement existing methods and processes [28]. For instance, patterns help to bridge the abstractions in the analysis and design phases with the concrete realizations of these abstractions in the implementation and maintenance phases. In the analysis and design phases, patterns also help to guide developers in selecting from software architectures that have proven to be successful. In the implementation and maintenance phases, they help to document the strategic properties of software systems at a level higher than source code and models of individual software modules.

According to Coplien [12], the term *software pattern* refers to the replicated similarity that enables variability and customization in each of the elements, i.e. dynamically adapting to fulfill changing needs and demands. He also states: *"Patterns aren't designed to be executed or analyzed by computers, (...), patterns are to be executed by architects with insight, taste, experience, and a sense of aesthetics"*.

In general, patterns have the following properties [27]: They

- can be abstract or concrete,
- can be a "blueprint" or template for design,
- have an inner structure and dynamics,
- document practice-proven "good" designs, architectures and abstractions beyond the granularity of single classes,
- define a common vocabulary and understanding of design principles,
- support the design of software with well-defined properties,
- help to manage complexity.

Further information about patterns can be found on the Internet at [1] and [4].

3.3 A Design Pattern Example: The Model View Controller Pattern

The model view controller (MVC) pattern [19] is probably the most widespread structuring principle for applications with a graphical user interface (GUI). It was introduced to reduce the development effort required for interactive systems, especially those which make use of multiple, synchronized presentations (views) of shared information. Examples for this pattern include graph packages displaying both bar-charts and pie-charts of the same data.

The aim of the MVC design pattern is to separate the application into a triad of components respectively objects (Figure 4):

- The application object (**model**), which holds the data values being manipulated and conducts all the computations and operations that can be applied,
- from the way it is represented to the user (**views**),
- from the way in which the user controls it (**controller**). The controller responds to all user actions and modifies the model and the views appropriately (*change*).

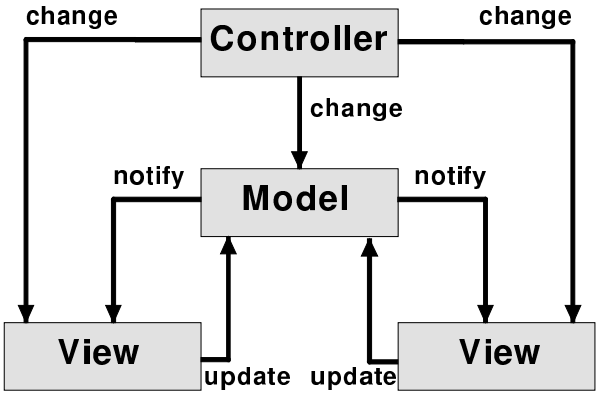

Figure 4: A graphical representation of the model view controller (MVC) pattern [34]

Thereby, the model knows nothing about the GUI, the manner in which the data values should be displayed, or the GUI actions that are necessary to manipulate the data.

The views refer to the model and use its methods to obtain (*update*) data they subsequently display. New views for the model can be created without rewriting it.

The controller knows about the interactions by which users manipulate data within the model, e.g. the controller object would receive mouse clicks and transform them into a call of the appropriate method of the model. In some situations the controller may also interact directly with a view without passing the model. Changes concerning the data of the model are immediately propagated to the views (*notify*).

The MVC pattern is an interface architecture pattern, i.e. not a low-level programming pattern. Due to space limitations, this (shortened) example of a design pattern has not been presented in a formal pattern notation (as it would have been correct, of course) but only for the purpose of giving at least one concrete example of a heavily used pattern.

3.4 Pattern Catalogs, Systems and Languages

Much of the existing literature on design patterns is organized as design *pattern catalogs* [18], [10] which present a collection of more or less independent solutions for common design problems.

A *pattern system* is a cohesive set of related patterns and adds deep structure, rich pattern interaction, and uniformity to a pattern catalog [4].

A *pattern language* is a collection of interrelated patterns, all sharing some of the same context, and perhaps classified into categories [21]. Compared with a pattern system, it adds robustness, comprehensiveness, and wholeness.

A pattern language includes rules and guidelines about order and granularity for applying and combining its patterns to solve a problem to large to be solved by any individual pattern. Pattern languages are not formal languages, though they do provide a vocabulary for talking about a particular problem [28].

There is a yearly conference called *Pattern Languages of Programming* (PLoP) and a book series called *Pattern Languages of Program Design* [11], [32], [22].

3.5 The Holy Grail of Design Pattern Writing

Patterns aim to improve communication by naming and concisely articulating the structure and behavior of solutions to common software problems. Thus, patterns help advanced developers to communicate domain knowledge, to learn new architectural styles or design paradigms, and guide inexperienced developers to avoid painful traps, pitfalls, and mistakes which were traditionally learned only by experience.

Good patterns capture proven empirical solutions that are not obvious. They also describe relationships, i.e. deeper system structures and mechanisms, not just abstract principles, strategies, theories or speculations [13].

A well written pattern should also have at least some of the following desirable qualities [4]:

- **Encapsulation and Abstraction**: Patterns serve as abstractions which contain and encapsulate domain knowledge and experience.
- **Openness and Variability**: Dynamically customizing and adapting to fulfill changing needs and demands.
- **Generativity and Composability**: Patterns may occur at different levels of conceptual granularity within the domain and may each lead to or be composed with other patterns at varying levels of scale.
- **Equilibrium**: A pattern should realize some kind of balance between its forces.
- **"Quality Without A Name" (QWAN):** The "quality without a name" brings "*incommunicable beauty and immeasurable value to a structure*" [4], i.e. some kind of subjective aesthetics.

So, the "holy grail" of pattern writing is a software engineering handbook in the form of a pattern language based on a collective compendium of "lessons learned" and "best practices" for solving known engineering problems. Engineers must then appropriately adapt the solutions mentioned in the handbook to achieve "*optimal trade-offs and compromises between known solutions, principles and constraints to meet the ever-increasing and ever-changing demands of cost, schedule, quality, and customer needs*" [4].

4 Comparison

The two languages depicted - features and design patterns - were developed more or less without influencing each other, but both aim at the overall goals of structured creation of objects, clear communication of requirements and easy re-use of results. To structure their tasks by meaningful elements they found their own vocabularies and languages, very often represented by graphical means for an easy reception by humans.

Both areas developed a system of catalogs where the engineers can lookup solutions to their problems which have already proven good or bad, and there they may also find explanations why. Those catalogs are an important building block in the never ending task of coding the domain knowledge of an engineering field, where the experience of thousands of people about the underlying process must be saved.

Due to the long experience of classical ME there is an enormously wide range of coded knowledge used as a matter of routine. Standards for components as machine and construction elements have reached a much higher degree of maturity than in SE, which allows ME to work in a stable network of subcontractors, while a software component industry with a network of manufacturers and suppliers of interoperable software components is still an issue.

One can find a handbook for every branch of classical ME, but encountering the challenges deriving from new and very complex work flows, ME lacks theory and methodology to cope with them. Everybody talks about concurrent and simultaneous engineering, but ME has no formal language to sufficiently describe and manage dynamic and parallel processes.

This is a field where SE is clearly in the lead, because handling processes and concurrent dynamic accesses to common resources have been treated by computer science since its early days. SE today has even reached a state where it is not only "able" to manage data, but also to use patterns for modeling processes and even the process of software development itself. The knowledge about - for example - multi-tasking operating systems that share resources and manage priorities with time constraints could possibly help ME to find its own process methodology.

There is also no clear ME methodology for the design with CAD systems and the maintenance of the resulting model data. ME has been looking upon CAD systems as just being tools for creating drawings for to long. Hence, CAD development is rather driven by computer graphics experts than by the needs of mechanical engineers, and the systems mainly deal with what you can see - which is pure geometry.

The feature approach wants to give back to the CAD system user the language with meaningful entities that he has been using in classical ME, but that is now enhanced with the possibility to pass those entities from one application to another within the product life-cycle.

There are mainly three approaches to how features can be used during the work on the product model, and all three have their analogy in the world of software design patterns.

- ***Design by features***

  Here the designer can use libraries of recurrent elements for creating a model where all its parts and regions carry some knowledge about their purpose, creation and further use. This procedure is analogue to modeling a software system using generative design patterns in a top-down approach.

- ***Feature recognition***

  This method is used to automatically identify meaningful entities in a purely geometric model without any additional information. Unfortunately this is a very hard task and far from being industrially applicable and the SE equivalence called *pattern mining* is also done by humans, not by computers.

- ***Feature mapping***

  As different views on the same model may lead to different meanings of the same underlying geometry and thus to different features, there is a need to transform features defined in different views into one another. The mapping process of converting for example a design feature into a manufacturing feature includes the most important ability of carrying the model data through the process without loss of information.

  There is no need for a *pattern mapping* mechanism in the SE process in the way that one pattern should be transformed into an other one although the same data structure might implement completely different design patterns.

  It is however quite common to interpret model data in different ways and to present it to the

user first as a table of numbers, second as a pie chart, and third as pictures of parts with different weight. One candidate (of several possible) for a design pattern realizing this concept is the above mentioned model view controller pattern as it provides a view-dependent but consistent model management.

The industrial realization of these three approaches are for ME a great step towards the goal of controlling the companies' work flows.

SE people have the same goal within their field, and the comparison between features and design patterns and their use shows a wide range of similarity. Design patterns are tools helping system analysts, architects, and designers to make their processes manageable.

We believe that feature technology could achieve the same in ME, and could also act as the link between ME and SE in order to describe, support, and manage the process of computer-aided product design. In fact, design patterns could be the means to describe features in a way that engineers of both fields can understand and talk about them (No more asking: *"What is a feature?"*).

5 Conclusion and Outlook

In this paper we showed the overall need for formal, simple, and flexible languages in the fields of software and mechanical engineering. Both worlds have developed – in a way of convergent evolution - their own solutions for the problem of capturing domain knowledge, easy re-use of results, and clear-cut communication.

The most recent computer-aided developments of these two languages are *features* and *design patterns*, which have been recognized as being similar in many ways. Nevertheless, there is still a missing communication link for mutual understanding between engineers of the two worlds, because they do not know each others' languages. We think that a combination of both – *design patterns for features* ("feature patterns") - could be that link and enable the engineers to work together more efficiently.

References

[1] *"Patterns Home Page"*, http://hillside.net/patterns/

[2] Alexander C., Ishikawa, S., Silverstein M.: *"A Pattern Language: Towns, Buildings, Construction"*, Oxford University Press, 1977

[3] Alexander C.: *"The Timeless Way of Building"*, Oxford University Press, 1979

[4] Appleton B.: *"Patterns and Software: Essential Concepts and Terminology"*, http://www.enteract.com/~bradapp/docs/patternsintro.html

[5] Bähr T., Weber C.: *"Neues aus dem Bereich der Feature-Technologie"*, CAD-CAM-REPORT Nr. 9, 1996, pp. 97-106

[6] Booch G., Jacobson I., Rumbaugh J.: *"The Unified Modeling Language, Documentation Set 1.1"*, September 1997

[7] Brown A. W.: *"Die Zukunft basiert auf Komponenten und Entwurfsmustern"*, in: OBJEKT spektrum 6/98, 1998, pp. 35-39

[8] Brown W. J., Malveau R. C., McCormick III H. W., Mowbray T. J.: *"AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis"*, Wiley, 1998

[9] Brunetti G., Rix J., Müller U.: *"Neues aus dem Bereich der Feature-Technologie II"*, CAD-CAM-REPORT No. 10, 1997, pp. 131-136

[10] Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M.: *"Pattern-Oriented Software Architecture - A System of Patterns"*, John Wiley and Sons, 1996

[11] Coplien J. O., Schmidt D. C.: *"Pattern Languages of Program Design"*, Addison-Wesley (Software Patterns Series), 1995

[12] Coplien J. O.: *"Software Design Patterns: Common Questions and Answers"*, in: Rising L., (Ed.), The Patterns Handbook: Techniques, Strategies, and Applications, Cambridge University Press, New York, January 1998, pp. 311-320

[13] Coplien J. O.: "Software Patterns", SIGS Books, New York, 1996

[14] Dankwort C. W., Janocha A. T., Podehl G.: *"Innovative Produktentwicklung - mit oder trotz Features"*, VDI-Tagung: "Features verbessern die Produktentwicklung - Integration von Prozeßketten", Berlin, January 20-21, 1997

[15] Dankwort C.W., Podehl G.: *"Industrial CAD/CAM Application and System Architecture - a Closed Loop"*, in: I. Horvath, K. Varadi (Eds.): TMCE '96, Proceedings of the International Symposium "The Tools and Methods for

Concurrent Engineering", Budapest, 29-31 May, 1996, pp. 206-211

[16] Darwin C.: *"The Origin of Species"*, Chapter 6: Difficulties on Theory, 1859

[17] Fowler M.: *"Analysis Patterns: Reusable Object Models"*, Addison-Wesley, 1997

[18] Gamma E., Helm R., Johnson R., Vlissides J.: *"Design Patterns : Elements of Reusable Object-Oriented Software"*, Addison-Wesley, October 1994

[19] Krasner G. E., Pope S. T.: *"A cookbook for using the model view controller user interface paradigm in Smalltalk-80"*, Journal of Object-Oriented Programming, 1(3):26-49, August/September 1988

[20] Krause, F.-L.; Vosgerau, F.H.; Yaramanoglu, N.: *"Using Technical Rules and Features in Product Modelling"*, in:Yoshikawa, H; Gossard, D. (Eds.): Intelligent CAD I, Elsevier, Amsterdam-London, 1987

[21] Lea D.: *"Patterns-Discussion FAQ"*, http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html

[22] Martin R. C., Riehle D., Buschmann F.: *"Pattern Languages of Program Design 3"*, Addison-Wesley (Software Patterns Series), 1997

[23] Meszaros G., Doble J.: *"A Pattern Language for Pattern Writing"*, http://hillside.net/patterns/Writing/patterns.html

[24] Mowbray T. J., Malveau R. C.: *"CORBA Design Patterns"*, John Wiley & Sons, 1997

[25] Ovtcharova J., Weber C., Vajna S., Müller U.: *"Neue Perspektiven für die Feature-basierte Modellierung"*, VDI-Z 139, No. 3, 1997, pp. 34-37

[26] Podehl G., Vajna S.: *"Durchgängige Produkt-modellierung mit Features"*, in: CAD-CAM-REPORT No. 3, March 1998, pp. 48-53

[27] Riehle D.: *"Entwurfsmuster für Software-werkzeuge. Gestaltung und Entwurf von Anwendungen mit grafischer Benutzungs-oberfläche"*, Addison-Wesley Longman, Bonn, 1997

[28] Schmidt D. C., Johnson R. E., Fayad M.: *"Software Patterns"*, in. Communications of the ACM, Special Issue on Patterns and Pattern Languages, Vol. 39, No. 10, October 1996

[29] Shah, J. J.: *"Conceptual Development of Form Features and Feature Modelers"*, Research in Engineering Design 2 (1990/91) 2, pp. 93-108

[30] Sunquist F.: *"Two Species, One Design"*, International Wildlife, September / October 1996, http://www.nwf.org/nwf/intlwild/serval.html

[31] Vajna, S., Wegner, B.: *"Features - Information Carriers for the Product Creation Process"*, in: Proceedings of the ASME Design Engineering Technical Conferences, Sacramento, September 14 - 17, 1997

[32] Vlissides J., Coplien J. O., Kerth N. L.: *"Pattern Languages of Program Design 2"*, Addison-Wesley (Software Patterns Series), 1996

[33] Weber C.: *"What is a Feature and What is its Use? - Results of FEMEX Working Group I"*, Proceedings of the International Symposium on Automotive Technology and Automation (ISATA), Florence, 1996, pp. 287-296

[34] Wheeler S.: *"Object-Oriented Programming with X-Designer"*, http://atddoc.cern.ch/Atlas/Notes/004/Note004-7.html#MARKER-9-5, 1996