2009

ADJUSTMENT STRATEGIES FOR MANAGING UNANTICIPATED CHANGES IN SOFTWARE DEVELOPMENT PROCESSES

Katja Andresen
University of Applied Science Berlin

Norbert Gronau
University of Potsdam

ADJUSTMENT STRATEGIES FOR MANAGING UNANTICIPATED CHANGES IN SOFTWARE DEVELOPMENT PROCESSES

Katja Andresen[1], Norbert Gronau[2]

Abstract

Software engineers face multiple challenges of managing unanticipated changes, dependencies, uncertainty, emerging demand patterns. In this contribution we focus on the process of software development and its design to especially cover unforeseen changes. The article presents a structural view on the (distributed) software engineering process introducing three domains that trigger adjustment opportunities of the engineering process. Hereafter the solution approach imposing the process model PEPMAD is outlined.

1 Introduction

The software industry is acknowledged worldwide as a key driver of social and economic growth [28]. Software Engineering has been a research field since the late 60's. The international conference on Software Engineering has been held regularly since 1975. Various meetings, events, journals and magazines are devoted to the topic of software engineering [28]. Nonetheless most modern approaches suffer from constraints that can be considered principles of the software engineering (as programming, testing, integrating). Of course, entire approaches emerged for which the motivation is to alleviate aspects. Nowadays advances address the building of self-adapting/managing/healing software systems [14]. Attempts are called automatic programming, constraint-based languages or aspect orientation to name a few **Fehler! Verweisquelle konnte nicht gefunden werden.**[19]. Hence new technologies are available to support the development of systems that respond more quickly and efficiently to a changing environment. However, most technology used to develop, maintain, and evolve software systems do not adequately cope with complexity, distribution and change [8], [16]. On the other hand, the capability to cope with changes is very often a unique selling proposition for software developing companies and essential for successful business.

In this article the focus was laid on the engineering *process* that is being affected when changes need to be managed. The results stem from the research project "IOSEW"[3] dedicated to explore the

[1] University of Applied Science Berlin
[2] University of Potsdam
[3] The research project IOSEW is funded by the Federal Ministry of Education and Research, Germany (IOSEW:01/SF 03).

link between flexibility strategies and (distributed) software product development. Four industrial partners that are small and medium sized contributed with specific software engineering processes and problems that provided the foundation for results. All of our industrial research partners develop and sell enterprise systems software.

This contribution is organised as follows: first a short overview of the principles in software engineering process are given. The process is divided into time depended phases introducing change qualities. Afterwards a structural process view is presented dividing the process into sub-views that contribute to run-time adjustment. Then requirements and challenges in terms of optimization are discussed before the solution approach is presented. The section on recent developments points at influential research work. Finally, the results are summarized.

2 Characteristics of the Software Development Process

The modern software paradigm requires basic activities. Although depths vary common initial planning activities comprise the definition of "ingredients" as engineering techniques such as specification, design, implementation et cetera. The selection of programming languages and basic project management methods are integral to modern software engineering. Along with that roles are assigned and a schedule is outlined in the initial planning process.
During the course changes occur that may or may not have been foreseen, e.g. additional customer requirements, resource fluctuation and more [7]. If all options are known in advance only anticipated changes are to be planned for. However, common software engineering activities show a different picture. *Table 1* provides typical adjustment challenges based on the research in IOSEW.

Table 1: Typical requirements to design for adjustments

Element	Challenges based on Case-Studies
A: Software development model	Unforeseen additional test phase
B: Software development model / Actor	Iteration of requirements analysis
C: Software development model	Combination of models, e.g. with development partners
D: Resource actor	Key-Developer leaves company
E: Environmental turbulence	Top-Down Management Decision to cancel cooperation with external division
F: Environmental turbulence	Management decision about cooperation with new partner

Design and Run-time considerations

The examples undermine; in the *design* phase of the software engineering process the need for adjustment can not be fully foreseen or anticipated. Thus, the planning process takes all known and foreseen constraints into consideration. However, the goal is to design for unanticipated changes that may occur within the dynamic context of the process. The process should be enabled to adapt to changes during *run-time*. The differentiation between design-time and run-time is common in various approaches dealing with flexibility issues in systems and software techniques. Especially software systems are oftentimes characterized in this form [1], [6].
In software systems design issues as bandwidth, sensors, represent parameter that can individually adapt to environmental influences embedded in control loop constructions for instance [4]. The view helps to link qualities as flexibility, fault-tolerance, tempo of adjustment with initial (planned

for) behaviour and dynamic real-time performance of the software system. On one hand we use this perspective to differentiate between initial software process design and on the other to evaluate dynamic process behaviour; here unanticipated adaptation comes into play as unforeseen process variations are of importance. We consider the following research questions that are further discussed in this paper:

1. How can the complex task of designing an adjustable software engineering process be divided into subtasks that are more easily to manage?
2. How can the system be enabled to adapt for best output during run-time when unanticipated changes need to be addressed (goal-function).
3. Along with that how can the responsiveness be evaluated. Is there an easy way to integrate concepts into daily business?

In the following section the software process is viewed thru the lens of increasing capability to adjust to changes.

3 Areas for Design and Run-Time Process Adjustment

Generally, the systems approach is taken characterizing an information system. *Figure 1* provides an unstructured overview of elements as the software development process, actors, organisational development and implementation techniques, relations between the arrangement of phases and the framework. The software (development) process itself is a structure typically comprised of the following steps: product and resource specification, development, implementation and release. For that a set of software development models exist describing approaches to activities that take place during the process. Actors as developers, members of the organisational and managing company structure as well as customers taking part in this process are involved.

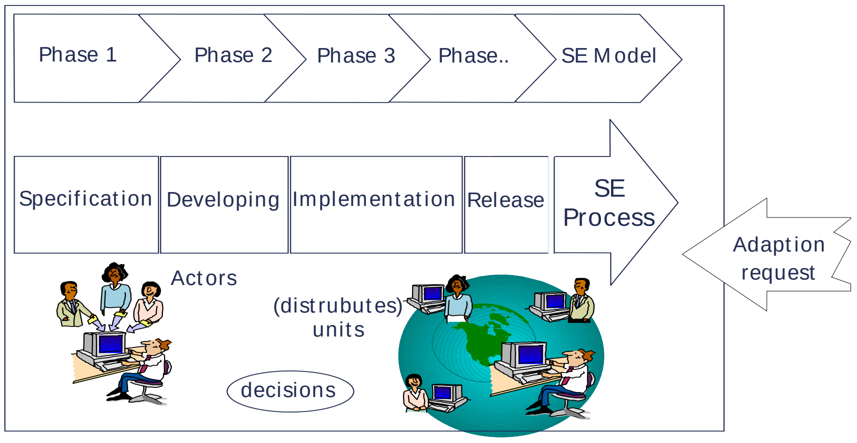

Figure 1: Elements of the Software Process

To classify software processes the following system description follows a three layer approach that structures domains to analyse and influence adaptive behaviour: 1) elements 2) structural topology, and 3) decision area.

1) *Elements* of the systems describe all entities that are part of the software-engineering process. Each entity can be seen as parameter that is somehow adaptable to changing conditions. The degree of adaptable behaviour may be given by nature, e.g. the capabilities of actors to cope with changes. Others are determined by rules like building blocks and coverage of the chosen software development models.

A few scenarios (*Table 1*) can be realized if elements as correcting variable are considered. The test team in 'A' could respond to lacking software quality by integrating additional steps. Mastering the loss of a key-developer in D, integrating an additional phase into the waterfall model leads to limitations of adjustments purely based on adjustment of elements. The latter examples require structural changes.

2) The *structural topology* addresses the links between elements. The links can be dynamically established, changed or simply dropped during run-time of the software development process. Links represent communication flows between actors; the interconnection of phases (first x then y), tasks and sub-tasks.
Process adjustments using structural topology require degrees of standardisation (as communication rules) to establish and deploy links. Also, modularity is a key to exchange and integrate elements. Roles therefore require clearly assigned profiles. Deploying modularity the process is structured into different stages to define variation points. Structural topology adjustment allows to completely revising the organisational set-up of the development process. However, elements (a phase, an actor) can be replaced without necessarily changing the structural topology.

3) The *decision area* refers to the (physical) distribution of elements. It relates to the area decisions to adapt are made. Distribution of elements raises the question of distributed (adaptable) process management. One of the features that are currently assigned to self-organizing systems is their decentralization [4]. Distributed adjustment involves further activities as distributed decision management and achievement strategies for desired global behaviour results. Decentralization of engineering processes may take different forms. Outsourcing, near- or off shoring are categories representing organizational aspects.
Mapping two or more sites requires interoperable processes and process models. To add or diminish locations scalable structures are needed.

Table 2: adjustment strategies (referring to scenarios in table 1)

Domain	Elements	Structural topology	Decision area
A: additional testing	- possible but requires scalability of se model, actor	- adjustment of se[4] model - model exchange - allocating additional work-load	- delegation of (modular) tasks, processes, phases - new partnering models
B: Failure in functionality	- possible but restricted to actor	- methods - implementation of additional requirement analysis	- (external) delegation - partnering as option
C: model combination in distributed environment	- requires structural change	- defining synchronization points - se model adjustment or exchange	- global (interoperable) process model - distributed software process management
D: key developer leaves	- Requires structural change; sd.-model might support profiles	- exchange of element - se model adjustment or - se model exchange	- delegation of activities
E: management decision – cancel cooperation	- requires structural change - work load increase possible (scalability)	- exchange or adjustment of elements	- reduction of decision area
F: management decision - cooperation	- requires structural change - work load decrease possible (scalability)	- se model combination	- expansion of decision area possilbe

[4] se model = software engineering process model (Waterfall, V Model, XP, RUP, …)

In the view of the abovementioned elements contain the properties to adjust to changes; each representing "parameters" similar to correcting variables in software technique and other disciplines. This form of adaptation does not change the structure but the value or status of the element (e.g. work-load).

As opposed to adjustment purely related to elements structural links allow a more dynamic behaviour towards unanticipated changes during run-time of the project. Common turbulences as loss and exchange of elements etc. require adjustments that address the structure or configuration of the se-process. Communication rules and behaviour are linked to this domain; likewise a revision of the (pre-planned) process model; the integration of redundant (work-load) absorbing structures.
The third domain covers partnering strategies (outsourcing of activities). Collaboration to mutually address the product development is linked with a shift from local or personal level to global level. Obviously, organisational changes as distributing product development, integrating new partner during the project stands for a high degree of flexibility during the project. However scenarios as merging locally distributed teams and process models involves building responsiveness, agility and adaptability into process architectures, project management methods, and organizations.

Adjustment Strategies and Requirements

Many sources of change are relatively unpredictable such as changes in leadership personnel (E, F in table 2), not specifiable (emerging) functions or events (B). Frequently, when product development goals are at risk engineers interact ad-hoc to then known facts [7]. In such cases the engineering model may not be suitable anymore. If documentation is suddenly crucial the waterfall model will be an option perhaps in exchange of less documentation focussed models. Hence expensive rework and uncertainty need to be accepted. The best way to narrow the problem space is to identify candidates reducing the consequences of unanticipated changes. At this point the domains of adjustment support deriving necessary properties:

Obviously, elements, links and decision space need to be *scalable* to handle quantitative aspects as work-load. Links between elements should be allowed to be established when needed. A scalable decision space adds and diminishes partners/sites that share the engineering problem.

Modular process structures defining sub-tasks and goals allow efficient modification of structures and decision space. Modular structures of engineering models ensure the definition of development phases and sub-tasks therefore exchange, distribution is an option if needed.

Interoperable structures ensure compatibility and standards [18]. At the level of elements role models (project manager, developer, tester etc.); communication rules; exchange formats, protocols and further aspects of standardization contribute to topology adjustment. The input and output services between (distributed) sites are also a matter of well defined standards and rules allowing efficient partnering.

Software engineering processes are considered *knowledge-intensive* processes. The skills of actors contribute in as much the engineering model may manage knowledge, e.g. contain best-practises (RUP model) or knowledge and skill profiles of actors.

Requirements that can be derived in order to respond and adapt to run-time dynamics relate to the context of elements, structural flexibility approaches but also se-process model adjustments. A selection has been illustrated. Understanding the factors that impact how process models are chosen, how they are developed, how they evolve and how they can be adapted are therefore critical topics for managerial attention.

Designing Distributed SE-Processes – Local versus Global Optimization

A key challenge for distributed SE-processes referring to the decision area is to balance local and global optimization (E, F *Table 2*).
In biologically inspired systems decentralization does not involve global coordination but local autonomy is postulated [12], [20]. Likewise actors as engineers, customers would act locally and the view is restricted to their immediate region. In terms of optimization this imposes a constraint upon the system (*Figure 1*). The question to ask is how a global process configuration can be achieved over *n* sites or engineering partners when commercial software is developed. Theoretically, each partner acts goal-oriented and is responsible for its own internal state and behaviour meaning by authority or contracting self-management is enforced.

Practically, organisational and geographical aspects stand for both complexity and variety in process and product design in proprietary models employed by commercial firms. The process optimization does strongly depend on how well actors are able to understand each other and find ways to mutually satisfactory results. So tensions and mistrust will cut off many options for joint benefit [7], [3]. Therefore, we suggest coming to an *achievable* solution that can be locally accepted.

Scenario E and F is an example for different levels of achievability. An expected top-down management decision to perhaps finish with a partner that serves as an equal engineering partner represents a very turbulent and unsecure environment for the (internal) team.
Central coordination, redundant structures rather than local autonomy appear more fruitful in the light of loosing product and engineering knowledge. From a local focus *redundancy* in structures is an option to adapt as the process benefits in future (e.g. test-algorithms, de-bugging, documenting etc.). Hereby distributed autonomy is reduced; central control structures are established; thus the focus is to reach a local optimum for the team afraid to lose a partner.
Based on the link between structure of product and organisations that participate in the development *structural analogies* represent the similarity between product and process design [11]. In other words decentralized processes require likewise product and process management structures for better adjustment (global optimization). Structural analogies are identified as key enabler for adaptability in related research [1], [7]. However, stability valuing trust and mutual benefit appear a basic must when designing distributed adaptable engineering processes that show overall-efficiency.

4 Solution Approach – PEPMAD

So far we have shown the enduring principles in software engineering; provided sub-views on three domains that trigger adjustment. We also showed our concept of achievability in terms of optimization. In this section managerial functions to support adaptive behaviour during run-time are outlined.
Adaptation during run-time responds to environmental context of the software engineering process, e.g. new customer demand patterns. Hence changes need to be registered. Afterwards a decision needs to be taken what way to go. Hence alternatives need to be found and evaluated, e.g. additional testing, partnering with X or Z etc. Third, to put the decision into action requires reconfiguring the

process. Therefore, to enable the software process to be more adaptable three basic functions are required: (1) context management; (2) option management and (3) configuration management. The context management gathers information about decision area, structural topology and elements. It corresponds to process monitoring (Figure 2). If changes are significantly unanticipated the adaptation management is engaged. The evaluation of alternatives is criteria based. The suitable alternatives are subject for reconfiguration activities. The criteria that serve for evaluation of alternatives are based on relevant properties (as scalability, modularity, interoperability, redundancy…) that need to be enhanced in order to adapt the domains of adjustment to design the process more adaptable.

The PEPMAD – Process Model

PEPMAD – the Potsdam Evolutionary Process Model for Adaptable Design is a process model (Figure 2). In general, a process model is a central outline for a systematic development of a system being an organisational framework. It specifies the order and the kind of actions between system elements in focus [13]. The goal of PEPMAD is to define features that are essential for a given software engineering process to be more adaptable during run-time.
The loop construction allows continuous course of activities and adaptation to new circumstances [27]. The evolutionary aspect is given by the integrated loop which implies self-optimization and continuous improvement [24]. The term applied to a software development system is a vision lent from biological context to underline the fact that the system itself has to recognize its needs and accordingly adapt its functions, retaining the useful and abolishing the disturbing elements [25]. Also, process adjustment is a continuous organizational task that should be integrated as soon as possible as software changes occur on the first day of development resulting in engineering process changes [8].

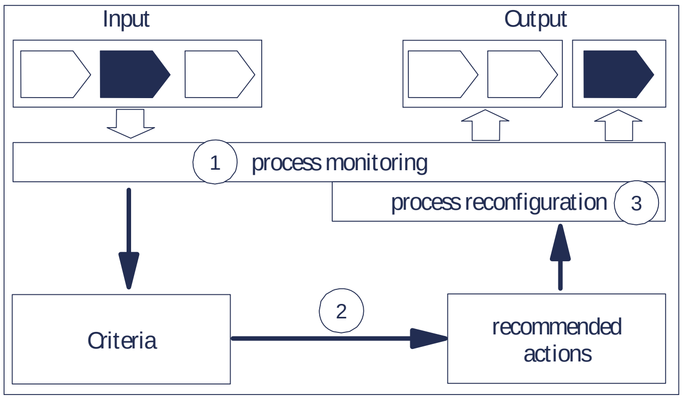

Figure 2: PEPMAD – Potsdam Evolutionary Process Model for Adaptable Design

PEPMAD @work

Applying PEPMAD there are two choices to make: the process or scenario to analyze and the depth of analysis. With regard to the first the differentiation between single site locations versus distributed (open) environment is of interest. Distributed processes need to be modular and interoperable to facilitate the work, to attract partners that understand the task and to contribute to it. By contrast, in single locations problems are more often solved by face-to-face communication. A

special stage represents the start-up engineering organisation that has just entered the market with a software product. At this point the benefits of modularity may not (yet) be of interest but the capturing of process and product knowledge to (iteratively) optimize project performance.
Based on the initial condition of the engineering process the evaluation of the decision area is linked with a weighting of important process qualities; for instance modularity and interoperability in distributed and others as self-optimization, knowledge management for start-up's. A catalogue of about 50 aspects captures the current situation. The model provides conclusions on this to support adjustment strategies. The situation-based weighting of the criteria does influence the importance and order of recommendations that are directly linked with questions.

An optional but recommended step especially in distributed environments is the communication analysis deploying KMDL® (Knowledge Modelling Description Language) to identify strategic positions in the communication network [15]. In addition to the questionnaire lived communication can be visualized that show established communication ways also bottle-necks. The following picture illustrates the PEPMAD contents (Figure 3).

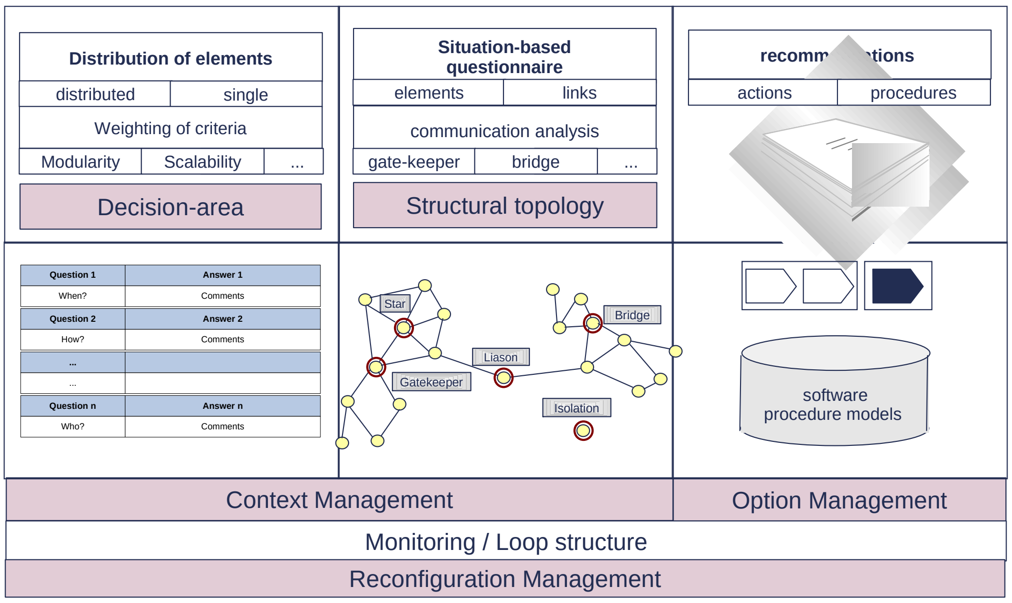

Figure 3: PEPMAD in detail

As said, the recommendations show options for better process adjustments. Options are directly linked with process models if they do contain the recommended activity. For instance, for a start up documentation may be recommended to ensure stability against loss of expertise; models supporting documentation as waterfall model would be listed. The idea is to additionally provide a choice of deployment packages that consist of established process models. Fifteen well-known process models have been pre-evaluated in terms of adaptation features. The best (most adaptable) results can be attained if the models can be integrated or combined to complete the recommendations.

5 Recent Developments

PEPMAD is being established within a research project "IOSEW". Software engineering processes of four software companies ranging from very small to medium sized companies provided the foundation for the process model. PEPMAD is tool-based covering the "simple run". For a first-time deployment three hours joining key personnel and consultant to ask and explain questions are necessary. Afterwards the simple cycle can easily be integrated into the business. The extended cycle deploying link analysis using KMDL® does require experts for data gathering and visualization; approximately 3 – 4 days are needed for a geographically distributed process between 2 to 3 partners.

The rest of the section points at a selection of related recent research that either influences or contributes to the results.
Autonomic computing (AC) presents autonomic elements that follow the well-known MAPE cycle: to monitor, analyze, plan and execute [22]. AC introduces a computer system. PEPMAD as said is a process model; nonetheless basic activities of the MAPE cycle are shared. PEPMAD in our opinion appears to be a special variation though the MAPE model is less specific.

A range of work has aimed to examine the link between a product's architecture and the characteristics of the organization that develops this product. The link in between is the process of production. The roots of this approach lie in the field of organization theory, where it has long been recognized that organizations must be designed to reflect the nature of the tasks that they must perform [23], [9].

In addition, discussions on flexibility concepts and methodologies, popular in the IT engineering world, termed agility, adaptation, changeability served as impulse though typically not agreed upon as they often reflect a specific level of consensus [1],[10].

6 Conclusions

Developers design software systems based on needs and constraints imposed by external factors. These constraints require organisations not only to adjust to them but also and especially to respond to them in shortest time.
We have investigated the approach to build a self-adapting software engineering *process* to manage turbulences during process run-time. The concepts fill the gap between structure and function of the engineering process. PEPMAD postulates a couple of main principles:

- Three domains of adjustments support overall process adjustment; namely elements, structural links and decision area.
- Based on well defined domains of adjustment the goal is to use them all during run-time by planning well in advance.
- To gather or buy information is one principle to diminish uncertainty.

Given these characteristics, PEPMAD would also benefit from research in several related areas. For example dynamic software product lines and product reuse [17], [25] , automatic versus distributed human decision making, and further structural analogy research [29].
Nonetheless to cope with dynamics each planning phase should be supported by process models that support change during project run-time taking possible adjustment strategies into consideration. PEPMAD is a blueprint to better adapt to the additional emergence of unanticipated change. The feasibility has been proven for small to medium software developing companies. To ensure broad utilization and adoption the process model would benefit from more case studies, which probably would contribute to refinements. Of interest the generalization of results for open-source software engineering processes remains uncertain. Additionally, new developments in process models, standards that influence the enduring principle of software engineering appear challenging.

7 References

[1] ANDRESEN, K.: Design and Use Patterns of Adaptability in Enterprise Systems, Gito, Berlin, 2006.
[2] ANDRESEN, K., GRONAU, N., Managing change – Determining the Adaptabiltiy of Information Systems, European Conference on Information Systems (EMCIS) 2006
[3] BALASUBRAMANIAM, R., LAN, C., ANNAN, M., PENG, X., Can distributed Software be Agile?, Communications of the ACM, 49, 41 – 46, 2006.
[4] BICOCCHI, N., MAMEI, M., ZAMBONELLI, F., Towards Self-organizing Virtual Macro Sensors, IEEE Computer Society, First International Conference on Self-adaptive and Self-organizing Systems (SASO), 2007.
[5] BLAIR G., S., COULSON, G., BLAIR L., DURAN-LIMON, H., GRACE, P., MOREIRA, R., PARLAVANTZAS, N., Reflection, self-awareness and self-healing systems, ACM Press, 9 – 14, 2002.
[6] BISHOP, J., HOORSPOL; N., Cross-Plattform Development: Software that Lasts, IEEE Computer, 39, 26 – 35, 2006.
[7] BOEHM, B.: Making a difference in the Software Century, IEEE Computer 41, 32 – 38, 2008
[8] BOHNER, S., An Era of Change-tolerant systems, IEEE Computer 40, 100 – 102, 2007.
[9] BURNS, T., STALKER, G., M., The Management of Innovation, Tavistock Publications, London, 1961.
[10] COALLIER, F., Standards, Agility, and Engineering, IEEE Computer 40, 100 – 102, 2007.
[11] CONWAY, M.E., How do Committee's Invent, Datamation, 14, 28-31, 1968
[12] CARILLO, L., MARZO, J.,L., VILA, P., MANTILLA, C., A., Ant Colonies for Adaptive Routing in Packet-Switched Communications Networks. In Eiben et. al. (editors) Parallel Problem Solving from Nature, 673 – 682, 1999.
[13] FITZGERALD, B., Formalised systems development methodologies: a critical perspective, Information Systems Journal, 6, 3 – 23, 1996.
[14] GHEIS; K., ULLAH KAHN, M., REICHLE, R., SOLBERG, A., Modeling of Component-Based Self-Adapting Context-Aware Applications for Mobile Devices, IFIP Working Conference on Software Engineering Techniques, 718- 722, 2006.
[15] GRONAU, N. FRÖMING, J., Eine semiformale Beschreibungssprache zur Modellierung von Wissenskonversionen. In: Wirtschaftsinformatik, 48, 349-360. 2006 (in German).
[16] GWANHOO, L., DELONE, W., ALBERTO-ESPINOSA, J., A., Ambidextrous Coping Strategies in Globally Distributed Software Development Projects, Communications of the ACM, 49, 35 – 40, 2006.
[17] HALLSTEINSEN, S., HINCHEY, M., SOOYOUNG, P.; SCHMID, K., Dynamic Software Procuct Lines, , IEEE Computer 40, 93 – 95, 2007.
[18] HANISCH, F., STRASSER, W.: Adaptability and interoperability in the field of highly interactive web-based courseware. In: Computers & Graphics, 27, 2003; 647-655
[19] HAREL, D., Can Programming Be Liberated, Period? IEEE Computer 41, 28 – 37, 2008.
[20] HERRMANN; K., WERNER, M., MÜHL, G., A Methodology for Classifying Self - Organizing Software Systems. International Transactions on Systems Science and Applications, 2 (1); 41 – 50, 2006.
[21] HIGHSMITH, J., COCKBURN, A., Agile Softwaer Development: the business of innovation, IEEE Computer, 34, 120 – 127.
[22] KEPHART, J. O., The Vision of Autonomic Computing, IEEE Computer 36, 41 – 52, 2003.
[23] LAWRENCE, P., R., LORSCH, J.W., Organization and Environment, Haward Business School Press, Boston, MA, 1967.
[24] LEHMANN, M., M., Feedback in the software development process, Information and Software Technology, 38, 681 – 686, 1996.
[25] MADAM HOMEPAGE, Mobility and ADaptation enAbling Middleware, http://www.intermedia.uio.no/confluence/display/madam/home (28.07.2008)
[26] MADHAVJI, N., H., The Process Cycle [Software Engineering], Software Engineering Journal, 6, 234 – 242, 1991.
[27] NUSEIBEH, B, Weaving together Requirements and Architectures, IEEE Computer, 34, 115 – 119, 2001.
[28] OSTERWEIL, L., J.; GHEZZI, C.; KRAMER, J.; WOLF, A.,L.: Determining the Impact of Software Engineering Research on Practice, Computer, IEEE Computer 41, 2008.
[29] V. HIPPEL, E., Task Partitioning: An Innovation Process Variable, Research Policy 19, 407 – 418, 1990.