

# Postmodern Knowledge Creation Approach in Software Product Development Companies

Majid Aramand

*Tampere University of Technology, majid.aramand@tut.fi*

## Abstract

This study applies the postmodern knowledge creation approach to the creation and development of knowledge in software product development companies and empirically tests the applicability of postmodern knowledge creation approach to software product design and development processes. The main characteristics of postmodern approach to knowledge creation are self-reflexivity in knowledge production, incredulity towards meta-narrative, capability of adaptation, and capability of being critical and suspicious of our own intellectual assumptions. It is hypothesized in this study that since the degrees of uncertainty and degree of change during software product design and development processes are relatively high, the characteristics of postmodern knowledge creation approach are applicable to the creation and development of knowledge in software product development companies. For testing the hypothesis, an empirical research is conducted on 52 small and medium-sized software product development companies in Canada. The findings of the empirical research support the main argument of this study and show that postmodern approach to knowledge is applicable to the creation and development of knowledge in most of surveyed software product development companies.

## Keywords

modernism, postmodernism, creation and development of knowledge, software product design and development companies

## Acknowledgements

I would like to thank software product development companies in provinces of Ontario and Quebec that participated in this research. I should also thank those who attended my presentation on this paper in the eBRF conference and made constructive and useful comments on different aspects of this study.

## Introduction

Software companies, especially software companies that develop software products for mass-market individual end-users are highly knowledge intensive organizations—and their competitive advantage significantly depend on their ability to create and develop ‘right’ and ‘useful’ knowledge. This knowledge partly consists of knowledge about software design and development techniques and methods, project management, and resource management—and partly consists of knowledge about company’s customers and market. The creation and the development of knowledge about customers and market in software companies usually takes place in a process of constant interaction, learning, and adaptation with company’s

functioning environment (customers and market). These processes of interaction and learning are the integral parts of a software company's product design and development methodologies and processes. The influences that these methodologies and process models have from either modern or postmodern scientific thoughts have impact on rightness and usefulness of knowledge that company creates and develops over time.

Modern approach to knowledge creation is based on having faith in individual and organizational rationalities and emphasizing on metanarrative and metamethod for interacting and understanding the functioning environment. In Postmodernism, however, we must possess the ability to be critical and suspicious of our individual and organizational rationalities towards our knowledge and understandings of the functioning environment. Postmodern approach to knowledge is basically characterized by incredulity towards metanarrative and metamethod, self-reflexivity in knowledge production, and capability of adaptation to the functioning environment.

This study is not attempting to introduce a new software design and development methodology or a method per se. The study aims at assessing whether the characteristics of knowledge creation and knowledge development in postmodernism are applicable to the creation and development of knowledge in software product development companies. In this paper, there is review on the theories of software product design and development methodologies, process models, and modern and postmodern approaches to knowledge. These subjects each encompass very broad body of literature. I have narrowed down the review to topics that are closely related to the main argument of the study. There is an empirical survey for testing the research hypothesis in this study. The survey was conducted as part of a broader study through an online survey on small and medium-sized software product development companies in Canada.

## Literature review

### Software product development processes

According to Cook and Wolf (1998), a software process is a set of activities applied to artifacts, leading to the design, development, or maintenance of a software system. Boehm and Ross (1988) suggest that the primary functions of a software process model are to determine the *order of the stages* involved in software development and evolution and to establish the *transition criteria* for progressing from one stage to the next. Examples of software processes include design methods, change-control procedures, and testing strategies, where the general intent of a software process is to coordinate individual activities so that they achieve a common goal (Cook and Wolf, 1998: 216).

A typical software development process framework is depicted in following figure. In this model, according to Hoch et al. (2000), at the project outset, during the 'upstream phase,' uncertainty is high, both in terms of the final outcome, as well as in the terms of schedule, cost, and other project parameters. As the project progresses, the uncertainty decreases and the project turns into the downstream phase. In that phase, the project is usually more stable, and uncertainty is lower and in fact, uncertainty decreases almost to zero at shipping or

project completion date. Maintenance and customer services at the tail end have relatively low uncertainty level.

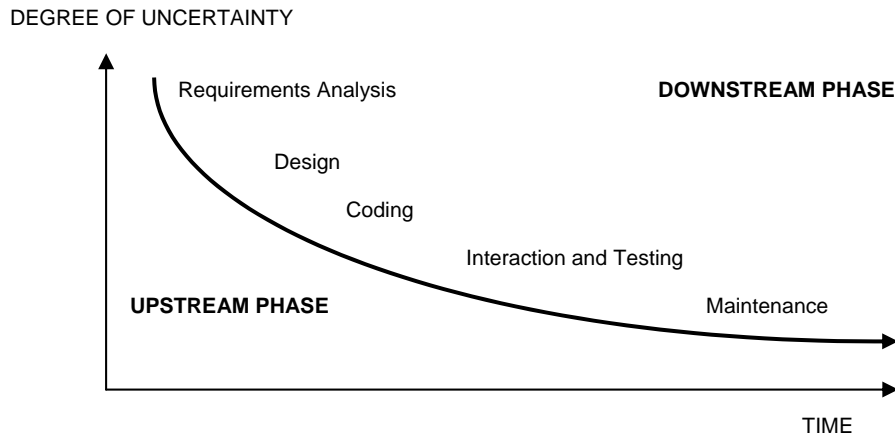


Figure 1. Upstream-downstream framework, adapted from Hoch et al., 2000

In the literature, a widely influential software process model is Capability Maturity Model (CMM), which was originally developed by the software Engineering Institute at Carnegie Mellon University. According to Paulk et al (1993), CMM was initially developed for providing software organizations with guidance on how to gain control of their processes for developing and maintaining software and how to evolve toward a culture of software engineering and management excellence. As shown in figure 2, CMM is characterized by five maturity levels that highlight the primary process changes made at each level (Paulk et al., 1993):

#### *Initial*

The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

#### *Repeatable*

Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

#### *Defined*

The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

#### *Managed*

Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

*Optimizing*

Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

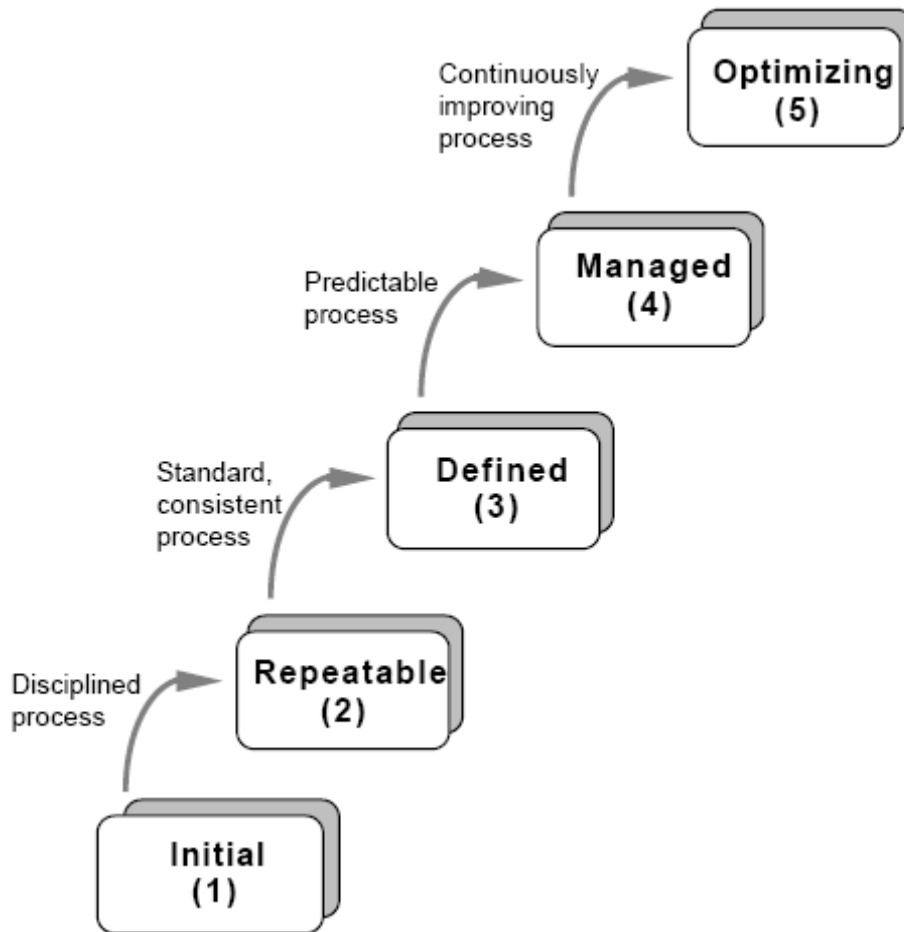


Figure 2. Five Levels of Software Process Maturity, adapted From Paulk et al., 1993

### Software development methodologies and methods

Boehm and Ross (1988) suggest that a software methodology or method differs from a software process model. They argue methodology/method's primary focus is on how to navigate through each phase (determining data, control, or "uses" hierarchies; partitioning functions; allocating requirements) and how to represent phase products (structure charts; stimulus-response threads; state transition diagrams).

Two methodologies are predominantly discussed in the literature of software design and development—plan-driven and agile methodologies. The term "plan" according to Boehm (2002) includes documented process procedures that involve tasks and milestone plans, and product development strategies that involve requirements, designs, and architectural plans. As depicted in following figure, Waterfall model is a typical plan-driven methodology, which consists of a set of phases, starting with system specification, with results cascading from one stage to another.

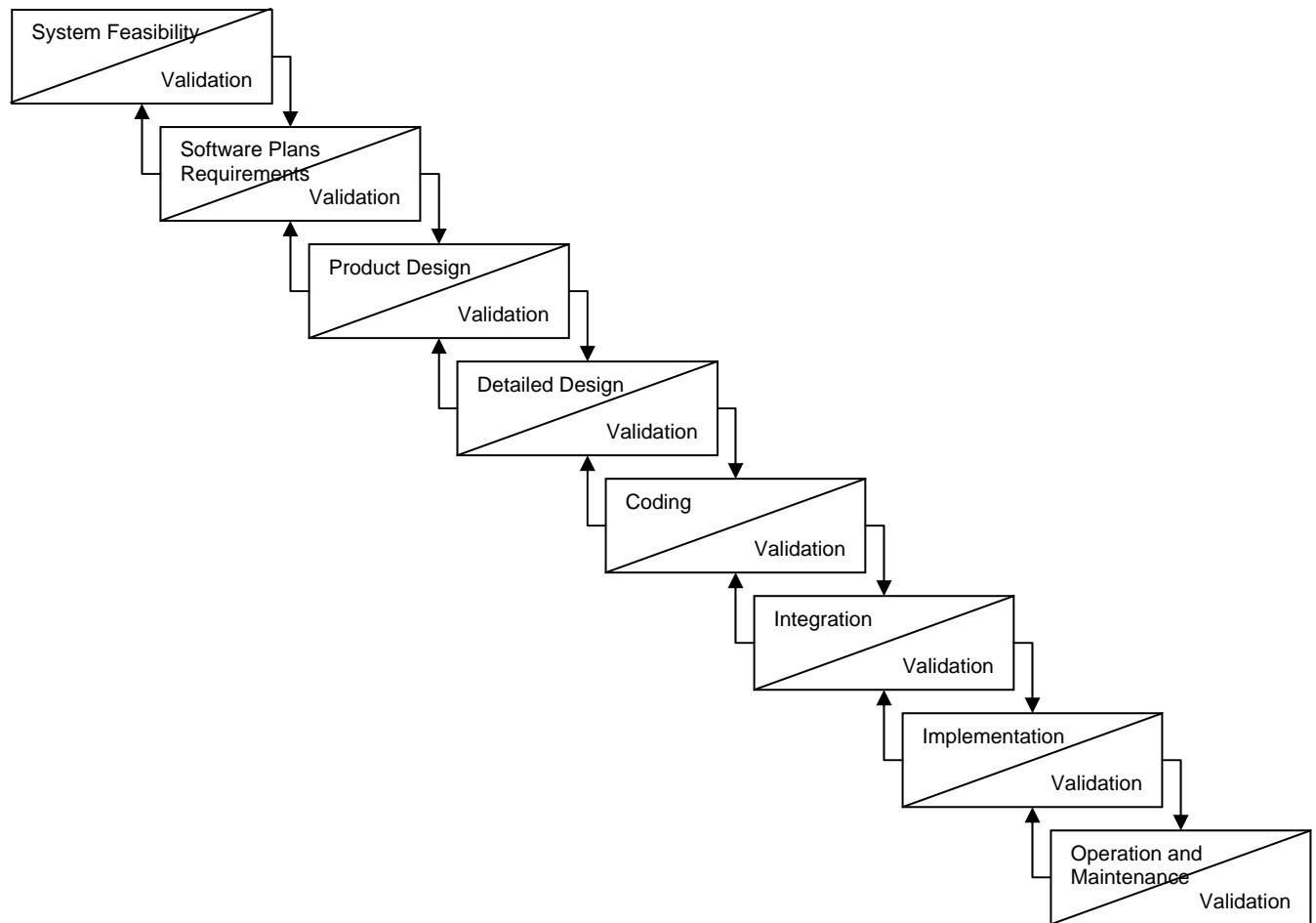


Figure 3. Waterfall model of the software life cycle, adapted from Boehm, 1998

Boehm (2002) argues that with requirements, plan-driven methods that emphasize heavyweight architecture and design documentation will encounter difficult to insurmountable problems in keeping up with rapidly changing requirements. Williams and Cockburn (2003) also in *agile software development: it's about feedback and change* explain that in the mid-1990s, both technology and the business environment kept shifting during the project, and both the requirements and project plans got out of date within even relatively short projects—customers became increasingly unable to definitively state their needs up front. They argue many companies and software developers realized that early planning and initial requirements documentation time consuming, expensive, frustrating, and in some cases, impossible to accomplish a project.

Due to the necessity of having a more flexible and dynamic software development methodology, a group of practitioners and researchers in Feb. 2001 met in Snowbird, Utah and introduced “agile” software product development methodology. They issued a manifesto called “Manifesto for Agile Software Development.” The key principals of agile manifesto include (Highsmith, 2002):

1. accepting changing requirements throughout the project
2. high priority is to satisfy the customer throughout the project
3. short delivery time and delivering working software frequently

#### 4. cooperation between business people and developers throughout the project

The major characteristics of an agile methodology as Abrahamsson et al. (2002) describe are being adaptive and responsive to the functioning environment, having incremental and continuous improvement in all design and development phases, having close communication and cooperation with customers and developers, and being simple and straightforward.

The agile methodology is applicable to several design and development methods, such as dynamic systems development (DSDM) and extreme programming (XP).

Plan-driven and agile software development methodologies both have pros and cons when are employed for designing and developing a software system. Boehm (2002) suggests that hybrid approaches that combine both methods are feasible and necessary for projects that combine a mix of agile and plan-driven home ground characteristics. He argues agile and plan-driven methods both form part of the planning spectrum and both have a home ground of project characteristics in which each clearly works best, and where the other will have difficulties.

## **Theoretical background**

### **Knowledge creation in modernism and postmodernism**

There is a broad consensus in the literature that modernist thought in the present century has important roots in philosophy of enlightenment—eighteen-century valorization of the individual mind that came to serve as the major rationalizing device for the twentieth-century (Cooper and Burrell, 1988; Gergen and Thatchenkery, 1996; Parker, 1992). According to Gergen and Thatchenkery (1996), the enlightenment was a historical watershed primarily owing to the dignity that it granted to individual rationality, where it was argued, within each individual lies a bounded and sacred principality, a domain governed by the individual's own capacities for careful observation and rational deliberation. They argue that the effects in the enlightenment thoughts were twofold: first, the individual mind of the worker, employee, and manager became a preeminent object of study, and, second, knowledge of the organization was considered a byproduct of the individual rationality of the scientific investigator. Parker (1992) also describes modernism as having faith in the power of the mind and in reason to a level at which it becomes equated with progress in developing knowledge to understand the nature—the world is seen as a system, which comes increasingly under human control as our knowledge of it increases.

If faith in reason and individual rationality are the cornerstone of modernism, in contrast, according to Power (1990), postmodernism stands for the 'death of reason.' Gergen and Thatchenkery (1996) argue that at the extreme, in postmodernist thought, the concept of individual rationality is found both conceptually flawed and oppressive in implications. Postmodernists argue that this is a form of intellectual imperialism that ignores the fundamental uncontrollability of meaning (Parker, 1992). The consequence of this is that we are advised to stop attempting to 'systemize,' 'define' or impose logic on events and instead to recognize the limitations of all our projects (Parker, 1992: 3) and try to possess the ability to be critical or suspicious of our own intellectual assumptions (Lawson, 1985).

Lyotard (1989) in *The Postmodern Condition: a Report on Knowledge* suggests that postmodernism can be understood as incredulity towards metanarratives (such as Marxism and structuralism) in contrast to modernism, which makes and appeal to just such narratives. He argues that knowledge under postmodern conditions can only be produced in “small stories” or “modest narratives,” mindful of their locality in space and time and capable of adapting or disappearing as needed. Rosenau (1992) also argues that in abandoning meta-narrative, postmodernists allow for renewed attention to the traditional and to the particular. Kilduff and Mehra (1997) in *postmodernism and organizational research* emphasize that postmodernism, above all, is eclectic rather than exclusive, in which it seeks to include and use techniques, insights, methods, and approaches from a variety of traditions, reaching backwards, forwards, and sideways with little regard for academic boundaries. From this postmodernist perspective, all styles are simultaneously available and it does not limit itself, then, to semiotic and deconstructive techniques, even though these approaches have been singled out as especially useful (Dickens and Fontana, 1994; cited by Kilduff and Mehra, 1997: 457).

### **Research argument and hypothesis**

As shown in the following figure 4, software solutions are usually in three customer solutions categories—custom-tailored software, embedded software and packaged mass-market software products. Custom-tailed software, as well as embedded software are usually designed and produced for fulfilling the requirements of a single commercial or industrial customer—usually in the forms of software projects. Packaged mass-market software products, however, are designed and intended for a large group of individual end-users.

The degree of customization (or degree of productization, according to Hietala et al., 2004) on the vertical axis is important for differentiating between software product and software project. The degree of customization ranges from standard “packaged” software products that are delivered “as is”, i.e., without any changes to a large number of customers, to embedded and customer tailored software, i.e., software that are developed according to the needs and specifications of individual customers (Hietala et al., 2004).

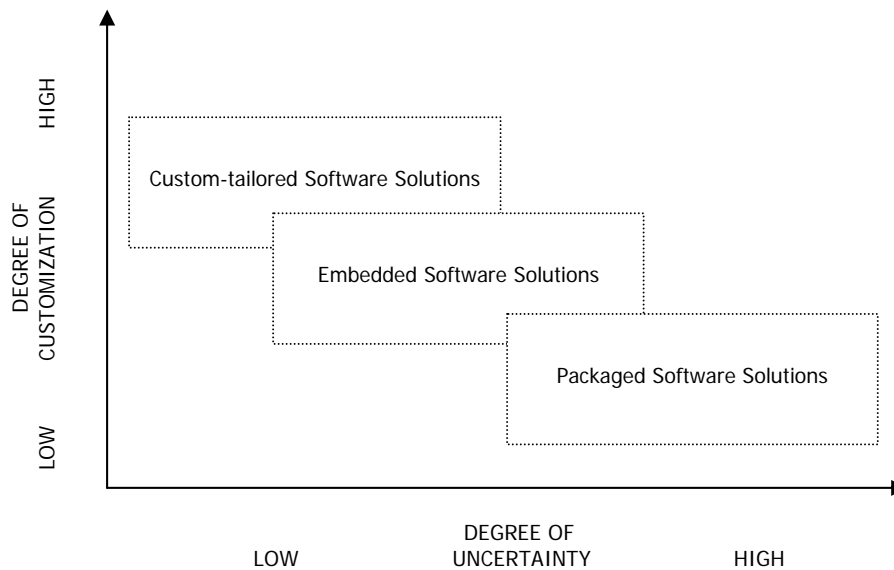


Figure 4. Software customer solutions categories

The horizontal axis in the following figure shows the degree of changes in design and development requirements during software design and development processes. In custom-tailored software solutions, software requirements are quite specific in the beginning of a software development project and they usually remain unchanged during the project development processes. In embedded software solutions, since the software solutions are designed for complementing hardware solutions, and in many cases, the degree of requirements are influenced by both end-users and operators, customer requirements are moderately specific and depending on the software project timeline, they have some changes in them. In packaged software solutions, if the software product is not new to the market, it usually becomes subject to fierce competition from the beginning of product design and development. This leads to uncertainty about end-users' needs and eventually many changes and uncertainties in design and development requirements throughout product development phases.

This study focuses on design and development in packaged software product development companies and not custom-tailored or embedded software development companies. The argument in this paper is that since the degree of uncertainty during software product development processes is relatively high, the characteristics of postmodern approach to knowledge creation and development are applicable to the creation and development of knowledge during software product design and development phases—and accordingly, I have driven the following hypothesis:

**Hypothesis:** *Postmodern approach to knowledge creation and development is applicable to the creation and development of knowledge in most software product development companies.*

As discussed before, the major characteristics of postmodern approach to knowledge are self-reflexivity in knowledge production, incredulity towards meta-narrative, capability of adaptation, and capability of being critical and suspicious of our own intellectual assumptions all the time. We can apply these characteristics to the creation of knowledge in software

product development companies. For instance, incredulity towards meta-narrative and capability of adaptation in postmodern approach to knowledge suggest that software product development companies during their product design and development phases do not follow the procedures of a particular methodology and they likely to change and switch their design and development methodologies and methods in order to adapt to the trends of changes in their functioning environment.

Few writings in the literature do emphasize that software product development companies use different design and development methods depending on their product development timeline. For instance, in *the innovator's dilemma*, Christensen (1997) points out that during a typical life cycle of products and services, different factors such as performance, reliability, convenience, and price become differentiators. Rifkin (2001) in *why software process innovations are not adopted* explains that depending on where you are in the cycle, you will select the method that optimizes the factor that is most important to your customers. He argues that you would select first a method that optimizes product performance, then later one that optimizes reliability, then later one that optimizes convenience, then finally one that optimizes cost. Arguably, this flexibility and tendency in selecting and using different design and development methods depending on project requirements and product development timeline reflects the very postmodern nature of the methodologies that are applied in these types of software development.

## Research processes and findings

The quantitative data for this research was gathered through an empirical research, which was part of a broader research on software companies aimed at studying the relationship between developing dynamic design capabilities and incorporating external factors into design and development processes. The target population of this research was small and medium-sized software product development companies. The sample was taken from 52 software product development companies in Canada. Companies in the empirical survey were randomly chosen from the Scott's Canada business directory. The sampling criteria were: 1) developing or developed a software product, 2) 10 to 250 employees, and 3) nature of software solution—software product developers (packaged software developers.)

The first sampling criterion was whether the company had been involved in a new product development project or not. Companies that had not had working experience in (new) product development projects were excluded from my target research group. The second sampling criterion was the size of the company. Data was collected from companies that had 10 to 250 employees. Companies that have less than 10 employees usually have informal organizational structure and lots of changes usually take place in these types of companies during a new product development project. Therefore, in these types of companies, it is difficult to measure design and development progresses properly. On the other hand, companies that have more than 250 employees, usually have several ongoing product development projects, where in this types of companies, it is also difficult to measure design and development progress of a single product development project. The third sampling criterion is the company's nature of product or solution. The hypothesis that I have driven in this paper is rather applicable to companies that are developing (have developed) packaged software products—not customized or embedded software products. Therefore, companies that were developing

customized software or embedded software were also excluded from my sampling research group.

For designing the online survey questionnaire, I employed the criteria of postmodern approach to knowledge creation and development that encompasses likelihood of change and adaptation in design and development methodologies during software design and development processes. I asked participant companies in the questionnaire that what type of software development methodologies they use and whether they change their design and development methodologies or not. I designed the questionnaire in a plain text format first, and pre-tested it in couple of companies for checking the wordings, smoothness, connectivity, and understandability of questions. Then I converted the questionnaire to ASP.NET format and again pre-tested it with couple of companies to check the quality and the functionality of the questionnaire's program. Also, at the end of the questionnaire, the participants were asked, if they had any comments or suggestions. These comments and suggestions were used for improving the user-friendliness of the questionnaire.

From 52 answers that I received from participant companies, 35 (67%) of them used a combination of several methodologies and methods or they did not use any particular methodology or method at all. Only 17 (33%) of total surveyed companies used particular methodologies and methods throughout their entire product development processes—and even these companies, mostly used agile software design and development methodologies (like XP and DSDM).

Throughout the survey, I also discussed about design and development methods in interviews and discussions that I had with product development and project managers. In my interviews, most managers did emphasize that they do not necessarily follow the procedures of a particular methodology or method and they tend to switch to different methods when they need to get a project done in a better way. The broad consensus in the surveyed software product development companies was “getting the job done in a cost effective and timely manner”—not necessarily pursuing the requirements of particular methods throughout the product development processes.

## Discussion

Most scholars who have worked on the relationship between philosophy of knowledge and the theories of software design and development methodologies agree to the view point that software development methodologies and methods have their roots in modernism and enlightenment discourses. For instance, in *postmodern software development* Robinson et al. (1998) argue that we have seen the development methods of software have been grounded in modernism and the enlightenment, but this has led to conflicts and contradictions, and a notion of ‘software crisis’ as the modernist metanarrative has broken down. Robert Filman (2005) in *postmodern software development* also suggests that in software development, the early emphasis on functional development yielded to structured programming, and, over the past 20 years, to the rise of computer science's modernism.

Waterfall life-cycle model (figure 3) and Capability Maturity Model (figure 2), which were discussed previously, are typical examples of methodologies and process models that are

highly influenced by modern scientific thoughts. The problem in software design and development methodologies like the Waterfall model is that they are not flexible and adaptive to the external changes during design and development phases. According to Robinson et al. (1998), one major problem with the modernist approach is problem of dynamism in the development of software, where requirements should be fully specified, up front. Also, according to Filman (2005), this type of development methodologies are concerned with efficient realism—it is hard to render even highly structured problems into code—programming is linear—things are said in the program tended to correspond, one to one, to things that happen when the program is executed.

In the literature there is a consensus that since agile methods focus on particular activities, such as programming and testing, rather than more esoteric activities such as analysis and design and because of their flexibility rather than a initial fixed plan, agile approaches have been characterized as “postmodern” when compared with traditional, “modern” software development methodologies based on forward planning and documentation (Cockburn 2001; Noble et al., 2004; Noble & Biddle 2002; Robinson et al., 1998). Designers and developers in agile software methods have constant self-reflexivity on their knowledge and understandings of customers, market, and design and development techniques and practices. As we saw previously, agile methods give priority to satisfying customers and accepting changing requirements throughout the product development phases. This emphasis on satisfying customers’ needs and accepting requirements’ changes requires constant evaluation of designers’ knowledge, and regular reflection on their assumption and understandings about customers’ needs and trends of market. Therefore, knowledge, skills, design capabilities, as well as software product(s) that are created through this process of constant self-reflexivity, will eventually become adaptive to the changes in the functioning environment.

Emphasis on modest or small and particular narrative in postmodern approach to knowledge is also applicable to agile software methods. In *Postmodern software development*, Robinson et al. (1998) suggest that software development has to become a more locally negotiated phenomenon. These local negotiations that they discuss about could be negotiation between designers and developers themselves or between designers and customers and users. In *agile software development methods*, Abrahamsson et al. (2002) point out that one of major characteristics of an agile method is small software releases with rapid cycles. For instance, extreme programming (which is an agile method) is based on the way the individual practices are collected (Abrahamsson et al., 2003: 245) and re-evaluated and integrated at each step of product development. Therefore, in agile methods, the emphasis is placed on the success and the outcome of individual practices, where these practices each are based on small and particular narratives. These particular narratives are essentially created through negotiations that have taken place in each step of product development between different parties.

As discussed before, the empirical findings of this study show that majority (67%) of surveyed software product development companies were using a combination of several methodologies and methods or they did not use any particular methodology or method at all. Even companies that used only one methodology or method throughout their entire product development processes mostly used agile software methodologies—that as discussed before, the postmodern approach to knowledge is more applicable to agile methods than the modern approach to knowledge. Therefore, since most surveyed software companies did not use a particular design and development method or they used a combination of agile methods

throughout their entire product development processes, we can argue that the research hypothesis is acceptable and postmodern approach to knowledge creation and development is applicable to the creation and development of knowledge in most software product development companies.

## **Conclusion**

Most software product development methodologies and process models (whether plan-driven or agile) have their roots in modern thoughts and modern approach to knowledge acquisition and knowledge creation. The reason of this stems from the fact that software is essentially a modern system, which functions on a hardware system like a computer, a mobile phone, a washing machine and etc. The very system-oriented nature of software has led to theatrical and philosophical development and conceptualization of software methodologies and process models as modern systems as well. On the other hand, software, particularly software products that are intended for individual end-users depend on high level of human involvements for their functionality and usability. This human-oriented nature of software functionality and usability requires more flexibility and adaptability of the software system to its functioning environment—and this flexibility and adaptability can only be incorporated into the software system when software design and development processes are taking place. This brings us to this conclusion that, although software is a modern system and modern way of thinking and conceptualizing play roles on the overall reliability and functionality of the software system, however, modern approach to knowledge that is based on faith in internal rationality (individual and organizational) is not an suitable knowledge creation and knowledge development approach for developing a software system with high degree of adaptability and user-friendliness. Therefore, a postmodern approach that is flexible, adaptable and critical about internal individual and organizational rationality during each step of software design and development processes is more suitable and applicable for the creation and development of knowledge in software product development companies.

The counterargument here is that even, if most software product development companies tend to change their design and development methodologies during product development processes, the criteria of postmodern approach to knowledge creation and development (adaptation to external causes) might not necessarily be the only or the main cause of this tendency. Other internal and organizational causes like funding uncertainties and organizational up-sizing or down-sizing might also play important roles in causing software companies to change their product design and development processes. There is no doubt that there are countless causes that might force a software company to change its design and development methodology throughout its product design and development phases. However, as long as a meta-narrative approach and dependability on a single methodology throughout the entire software product design and development phases is not working and software company has to switch to a different methodology time-to-time, we can argue that what ever the causes of changes are (whether internal or external), a postmodern approach to knowledge creation and development is more suitable for assessing the creation and development of knowledge in software product development companies than a modern approach.

## Need for further research

There is a need for further theoretical and empirical research on applicability of postmodern approach to knowledge creation and development in agile software design and development methodology and methods.

## References

- Abrahamsson P, Korkala M. 2004. Extreme Programming: Reassessing the Requirements Management Process for an Offsite Customer. Proceedings of 11th European Conference on Software Process Improvement Trondheim, Norway November 10-12: 12–22.
- Abrahamsson P, Salo O, Ronkainen J, Warsta J. 2002. Agile software development methods. VTT Technical Research Centre of Finland Oulu VTT Publications 478.
- Abrahamsson P, Warsta J, Siponen MT, Ronkainen J. 2003. Technical papers: software process: New directions on agile methods: a comparative analysis. IEEE Proceedings of the 25th International Conference on Software Engineering Portland, Oregon May 3-10: 244–254.
- Abrahamsson P. 2003. Extreme programming: first results from a controlled case study. IEEE 29th Euromicro Conference (EUROMICRO'03) Belek-Antalya, Turkey Sep. 01–06: 259–266.
- Adams GL, Lamont BT. 2003. Knowledge management systems and developing sustainable competitive advantage. *Journal of Knowledge Management* 7(2): 142–155.
- Ancori B, Bureth A, Cohendet P. 2000. The economics of knowledge: the debate about codification and tacit knowledge. *Industrial and Corporate Change* 9(2): 255–287.
- Anderson JC, Gerbing DW. 1982. Some methods for respecifying measurement models to obtain unidimensional construct measurement. *Journal of Marketing Research* 19(4): 453–460.
- Bamberger J. 1997. Essence of the capability maturity model. *Computer* 30(6): 112–114.
- Banerjee N, Bhattacharya S. 2002. Creating an agile software development organization: a key factor for survival in today's economy. *IEEE International Engineering Management Conference* 1: 230–233.
- Boehm B, Abts C, Chulani S. 2000. Software development cost estimation approaches—A survey. *Annals of Software Engineering* 10(1-4): 177–205.
- Boehm B. 2000. The Spiral Model as a Tool for Evolutionary Acquisition. *Software Engineering Institute Spiral Development Workshop* February 9: 1–34.
- Boehm B. 2002. Get ready for the agile methods, with care. *Computer* 35(1): 64–69.
- Boehm BW, Ross R. 1989. Theory-W software project management principles and examples. *IEEE Transactions on Software Engineering* 15(7): 902–916.
- Boehm BW, Sullivan KJ. 2000. Software economics: a roadmap. *ACM Proceedings of the Conference on The Future of Software Engineering* 319–343.
- Boehm BW. 1988. A spiral model of software development and enhancement. *IEEE Computer* 21(5): 61–72.
- Burrell G. 1988. Modernism, Post Modernism and Organizational Analysis 2: The Contribution of Michel Foucault. *Organization Studies* 9(2): 221–235.
- Burrell G. 1994. Modernism, postmodernism and organizational analysis 4: The contribution of Jurgen Habermas. *Organization Studies* 15(1): 1–19.
- Calas MB, Smricich L. 1999. Past Postmodernism? Reflections and Tentative Directions. *Academy of Management Review* 24(4): 649–671.
- Carlsson SA. 2003. Knowledge managing and knowledge management systems in inter-organizational networks. *Knowledge and Process Management* 10(3): 194–206.
- Carmel E, Becker S. 1995. A process model for packaged software development. *IEEE Transactions on Engineering Management* 42(1): 50–61.
- Christensen C. 1997. *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail*, Harvard Business School Press, Boston, Mass.

- Cockburn A, Highsmith J. 2001. Agile software development, the people factor. *Computer* 34(11): 131–133.
- Cockburn A, Highsmith J. 2001. *Agile Software Development*. Addison Wesley Longman. Online Edition: nkcho.isp.st.
- Cockburn A. 2000. Learning From Agile Software Development–CROSS TALK. *The Journal of Defense Software Engineering Part One*: 10–14.
- Cooper R, Burrell G. 1988. Modernism, Postmodernism and Organizational Analysis: An Introduction. *Organization Studies* 9(1): 91–112.
- Cooper R. 1989. Modernism, Post Modernism and Organizational Analysis 3: The Contribution of Jacques Derrida. *Organization Studies* 10(4): 479–502.
- Corno F, Reinmoeller P, Nonaka I. 1999. Knowledge Creation within Industrial Systems. *Journal of Management & Governance* 3(4): 379–394.
- Donaldson L. 2003. A Critique of Postmodernism in Organizational Studies. *Postmodernism and Management: Pros, Cons, and the Alternative*. Edited by: Locke EA. Elsevier Science LTD. *Research in the sociology of organizations* 21: 169–202.
- Drew DW. 1992. Tailoring the software engineering Institute's (SEI) Capability Maturity Model (CMM) to a software sustaining engineering organization. *Proceedings of Conference on Software Maintenance Nov. 9-12*: 137–144.
- Edge H. 2001. Readings in the Philosophy of Science: From Positivism to Postmodernism. *The Journal of Parapsychology* 65(1): 96–102.
- Featherstone M. 1991. *Consumer culture and postmodernism*. Sage Publication: London, UK.
- Filman RE. 2005. *Postmodern Software Development*. IEEE Computer Society 4-6.
- Gergen KJ, Thatchenkery TJ. 1996. Organization Science as Social Construction: Postmodern Potential. *Journal of Applied Behavioral Science* 32(4): 356–77.
- Ghate O. 2003. Postmodernism's Kantian Roots. *Postmodernism and Management: Pros, Cons, and the Alternative*. Edited by: Locke EA. Elsevier Science LTD. *Research in the sociology of organizations* 21: 227–245.
- Grandori A, Kogut B. 2002. Dialogue on organization and knowledge. *Organization Science* 13(3): 224-234.
- Grant RM. 1996. Toward a knowledge-based theory of the firm. *Strategic Management Journal, Summer Special Issue* 17: 109–122.
- Hanna M. 1995. Farewell to waterfalls? *Software Magazine* 15(5): 38–46.
- Hassard J, Kelemen M. 2002. Production and Consumption in Organizational Knowledge: The Case of the Paradigms Debate. *Organization* 9(2): 331–56.
- Hassard J. 1993. *Postmodernism and organizations*. Sage Publications: London, UK.
- Hassard J. 1993. *Sociology and organization theory: positivism, paradigms and Postmodernity*. Cambridge University Press: Cambridge, UK.
- Hassard J. 1994. *Postmodern Organizational Analysis: Toward a Conceptual Framework*. *Journal of Management Studies* 31(3): 303–324.
- Hietala J, Kontio J, Jokinen JP, Pyysiäinen J. 2004. Challenges of Software Product Companies: Results of a National Survey in Finland. *IEEE Computer Society, The 10<sup>th</sup> International Software Metrics Symposium, Chicago, U.S.A, Sept 14-16*.
- Highsmith J. 2002. *Agile Software Development Ecosystems*, Addison-Wesley Professional.
- Jacques R. 1992. Critique and Theory Building: Producing Knowledge “from the Kitchen.” *Academy of Management Review* 17(3): 582–606.
- Jacques R. 1997. Classic Review: The Empire Strikes out: Lyotard's Postmodern Condition and the Need for a “Necrology of Knowledge.” *Organization* 4(1): 130–142.
- Kallinikos J. 1997. Classic Review: Science, Knowledge and Society: The Postmodern Condition Revisited. *Organization* 4(1): 114–129.
- Kilduff M, Mehra A. 1997. Postmodernism and organizational research. *Academy of Management Review* 22(2): 453–481.
- Kilduff M. 1993. Deconstructing Organizations. *Academy of Management Review* 18(1): 13–31.

- Knights D. 1997. Organization Theory in the Age of Deconstruction: Dualism, Gender and Postmodernism Revisited. *Organization Studies* 18(1): 1–19.
- Kruchten P. 2005. Software Design in a Postmodern Era. *IEEE Software* 22(2): 16–18.
- Kukla A. 1997. Beyond Positivism and Relativism: Theory, Method, and Evidence. *The British Journal for the Philosophy of Science* 48(3): 447–454.
- Laudan L. 1990. Normative Naturalism. *Philosophy of Science* 57(1): 44–59.
- Lyotard JF. 1989. *The postmodern Condition: A report on Knowledge*. University of Minnesota Press: Minneapolis, MN.
- Mahoney MS. 2004. Finding a history for software engineering. *IEEE Annals of the History of Computing* 26(1): 8–19.
- McKinley W. 2003. Postmodern epistemology in organization studies: a critical appraisal. *Postmodernism and Management: Pros, Cons, and the Alternative*. Edited by: Locke EA. Elsevier Science LTD. *Research in the sociology of organizations* 21: 203–225.
- Nodoushani O. 2000. Epistemological foundations of management theory and research methodology. *Human Systems Management* 19(1): 71–80.
- Nonaka I, Takeuchi H, Umemoto K. 1996. A theory of organizational knowledge creation. *International Journal of Technology Management* 11(7-8): 833–846.
- Nonaka I. 1991. The knowledge-creating company. *Harvard Business Review* 69(6): 96–104.
- Osterweil L. 1987. Software processes are software too. *ACM Proceedings of the 9th international conference on Software Engineering*.
- Osterweil LJ. 2003. Understanding process and the quest for deeper questions in software engineering research. *ACM SIGSOFT Software Engineering* 28(5): 6–14.
- Parker M. 1992. Post-Modern Organizations or Postmodern Organization Theory? *Organization Studies* 13(1): 1–17.
- Paulk MC, Curtis B, Chrissis MB, Weber CV. 1993. Capability maturity model, version 1.1. *IEEE Software* 10(4) 18–27.
- Ravichandran T, Rai A. 2003. Structural analysis of the impact of knowledge creation and knowledge embedding on software process capability. *IEEE Transactions on Engineering Management* 50(3): 270–284.
- Redmond JA, Ryan KT. 1987. Prospects for a Method-Driven Software Development Environment. *Information and Software Technology* 29(8): 421–430.
- Robinson H, Hall P, Hovenden F, Rachel J. 1998. Postmodern Software Development. *The Computer Journal* 41(6): 363–375.
- Rudel TK, Gerson JM. 1999. Postmodernism, institutional change, and academic workers: A sociology of knowledge. *Social Science Quarterly* 80(2): 213–228.
- Salo O, Abrahamsson P. 2004. *Empirical Evaluation of Agile Software Development: The Controlled Case Study Approach*. Springer-Verlag Berlin Heidelberg 3009: 408–423.
- Srinivasan K., Fisher D. 1995. Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering* 21(2): 126–137.
- Tiedeman MJ. 1990. Post-mortems-methodology and experiences. *IEEE Journal on Selected Areas in Communications* 8(2): 176–180.
- Wiklund J, Shepherd D. 2003. Knowledge-based resources, entrepreneurial orientation, and the performance of small and medium-sized businesses. *Strategic Management Journal* 24(13): 1307–1314.