Agile game development: lessons learned from software engineering

Rens Kortmann
Casper Harteveld

Delft University of Technology
Faculty of Technology, Policy, and Management
P.O. Box 5015
NL-2600 GA Delft
The Netherlands
l.j.kortmann@tudelft.nl
c.harteveld@tudelft.nl

Keywords: game design, project management, agile development, software engineering, triadic game design, simulation games

Abstract

This paper proposes an agile model to develop games that simulate professional practices and policy evaluation. During the last decades, agile development models have evolved in a related design science, *viz.*, software engineering, to increase the quality and efficiency of large software development projects. Taking into account the similarities and dissimilarities between game development and software development, we show a first step towards an agile game development model based on a formal design approach called Triadic Game Design (TGD). We further devised a workshop to initiate an agile game development process during the first phase of a game design project.

1 Introduction

Many games that simulate professional practice or policy evaluation are the result of long, costly development projects. Regularly, the agreed project duration and budget are exceeded (see, for instance, McHaney, White, and Heilman (2002) for a survey of simulation project success and failure). Moreover, the quality of these games often does not meet the clients' expectations. In this paper we propose an agile game development model that may overcome some of these problems.

Although there exists preciously little literature that evaluates game development processes, we were able to derive the following four possible causes from communication with practitioners and personal experiences:

- The *technical complexity* of certain games has increased in recent years: web-based games, games in 3D virtual worlds, etc.
- *Management* of game design projects is often poor and ad hoc
- Clients often have unrealistically high *expectations* of projects
- At the same time, clients are often not well able to formulate the project goals and the product requirements which leads to project *uncertainty*

Considering the developments and problems sketched above, we may wonder if the existing game development models are well enough equipped to deal with them. How can we improve the existing models to support game development projects? And how can we improve the application of these models by project managers? In this paper we investigated the first question by drawing lessons from another design science, namely

software engineering, which faces similar problems with technological complexity, project management, client expectations and uncertainty (Wysocki, 2006). After this, we looked into how these lessons could be incorporated into a feasible game development model.

The paper is structured as follows. In the following section, recent developments in software engineering are scrutinised to reveal some valuable lessons for game development. In Section 3 our approach to agile game development, based on a game design approach called *Triadic Game Design*, is demonstrated. Finally, Section 4 draws conclusions and suggests further work.

2 Lessons from software engineering: agile development

Software engineering is a design science that provides methods for the development, implementation, and maintenance of software products (cf., Pfleeger & Atlee, 2005). When reduced to its bare essentials, as we will demonstrate, software engineering shows great resemblance to other design sciences, such as game design. Therefore, it seems worthwhile to look for good practices in software engineering and try and use them to improve existing game design methods.
Software engineering has undergone major changes in the past years. The discipline has shifted from a linear "waterfall" approach to a, what is commonly called, "agile" approach to software development (cf., Hazzan & Dubinsky, 2008; Wysocki, 2006). In this section, we compared software engineering to game design and demonstrate the possible benefits of applying an agile approach to game design.

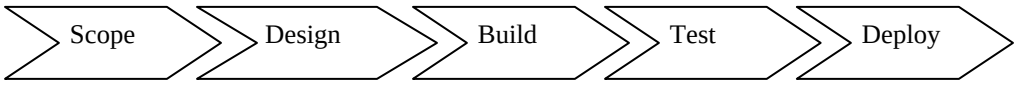
Software engineering vs game design

A considerable overlap exists between the design phases used in game design and those in software engineering. To demonstrate this point in more detail, we compare a widely accepted approach to game design (Duke, 1974; Duke & Geurts, 2004) to the traditional design phases identified for software engineering (Wysocki, 2006).

Figure 1 Five game design phases presented by Duke (1974) and Duke and Geurts (2004)

In the figure above, Duke and Geurts distinguish five phases for game design. Every phase consists of several steps, 21 altogether. In phase 1, five steps lead from project organisation to a set of game specifications. Thereafter, in phase 2, the underlying system is analysed and the project scope is negotiated with the client. A blueprint of the game is created in phase 3, consisting of five steps to select and define design specifications, for instance, game elements, techniques, a format, and a way to report on the design. During phase 4, a prototype is built, tested, modified and technically evaluated until the game is considered finished. Eventually, in phase 5, the game is integrated in the client's environment, facilitated, and disseminated. A final report is drawn up to properly close the project. For a more in-depth explanation of the 21 design steps, we refer to the original publications (Duke, 1974; Duke & Geurts, 2004).

Figure 2 Five design phases of software engineering (Wysocki, 2006)



In figure 2, five phases are distinguished that make up the design process of software engineering according to Wysocki (2006). The scheme above perhaps slightly oversimplifies the complexity of software development, but is nevertheless useful in conveying the message of this section – to draw lessons from innovative software engineering approaches and apply them to game design. The five phases are explained below and it is shown how they relate to the game design process in figure 1.

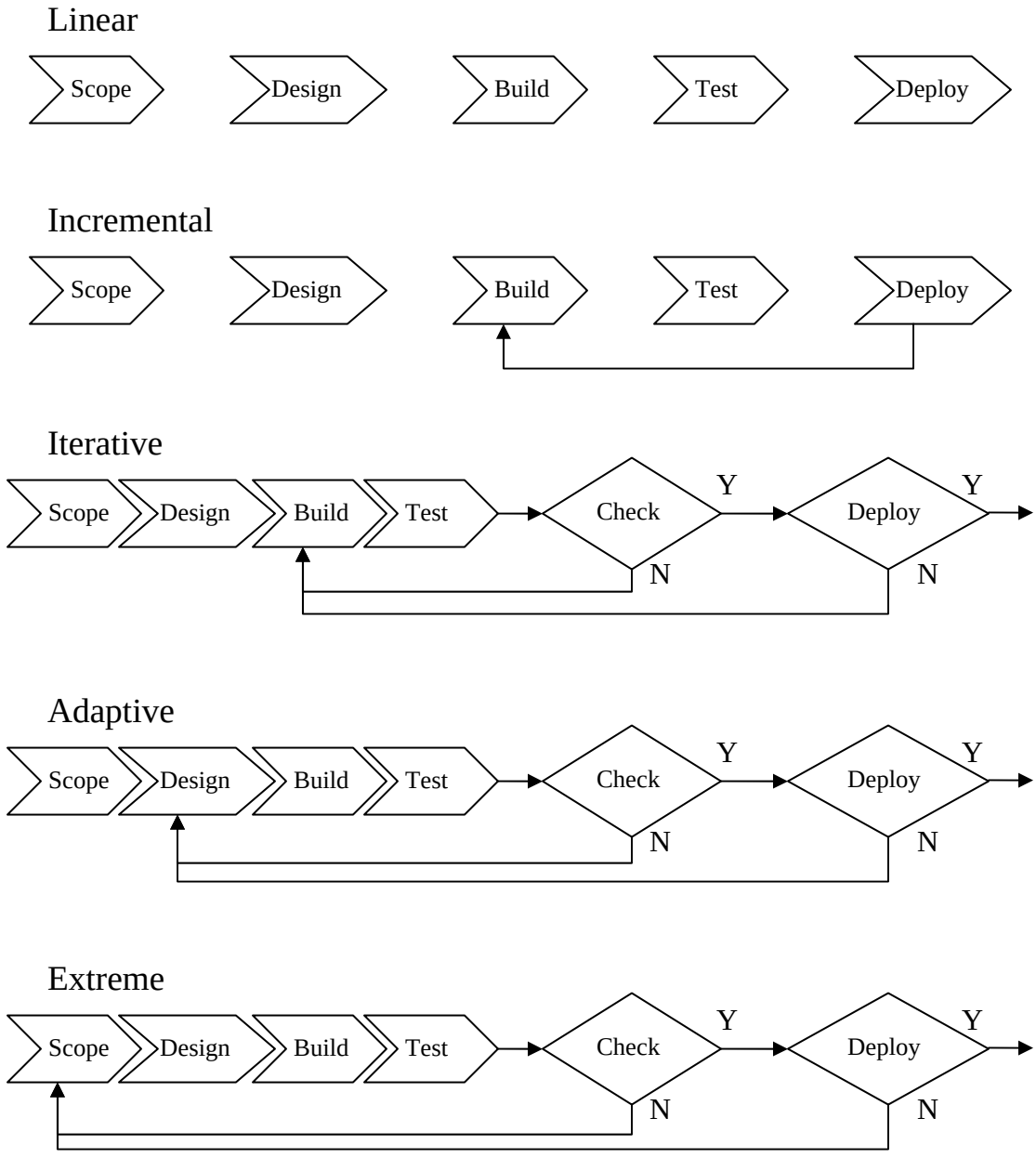
- **Scope** In the scope phase, the software project is initialised: the organisation, scope, and business case are determined, requirements and constraints are listed, a high-level planning is devised, and all is documented in a project overview statement. This phase roughly coincides with phases 1 and 2 in game design, with the exception that game designers focus more on understanding the underlying socio-technical system around which the game is built.
- **Design** The second phase comprises the design of the system and a detailed conceptual design, which are written down in a software design document. In this phase, a functional description of the artefact is sketched based on the requirements and constraints formulated in phase 1. The functional systems design is then elaborated in detail to prescribe the system architecture, coding techniques, and use of existing tools and libraries. This phase roughly overlaps with game design phase 3 (designing the game).
- **Build** The building phase consists of the actual software coding needed to construct the artefact. Here, it is decided how the prescribed architecture, techniques, etc., from phase 2 are implemented as programming constructs using one or multiple programming languages and one or multiple hardware platforms. This phase is comparable to the first part of phase 4 of game design (developing the game).
- **Test** The fourth phase in software engineering is devoted to testing the code in parts and the system as a whole: verification (does the software system comply with its specifications and constraints?), validation (does the system accomplish its intended requirements?). User testing: usefulness (which tasks can be performed with the system), usability (easy of use). Although Duke and Geurts emphasise the importance of play-testing, their general design model does not accommodate a separate design phase for testing as is the case in software engineering. Instead, in the game design process shown above, testing is part of the development phase.
- **Deploy** Finally, the fifth design phase in software engineering is marked for deployment of the system: implementation in the organisation, integration with other systems, development of manuals, end-user training, etc. This phase resembles phase 5 in game design.

Although game design is not the same as software design, it may be concluded from the comparison above that the design phases underlying game design and software engineering show great resemblances. In fact, many design sciences share the five phases mentioned. For this reason, it is expected that modern software engineering methods may provide helpful insights for game design as well.

Software development models

When the five phases are incorporated into a process flow diagram, we obtain a software development model. In Figure 3 we discuss five relevant software development models (adopted from Wysocki, 2006) and draw lessons from them for game development. Each model is based on the five phases presented in figure 2. For an in-depth analysis and explanations of the specific models mentioned we refer to the original publication.

Figure 3 Five software development models (adopted from Wysocki, 2006)



- **Linear** The linear development model consists of five dependent phases that are executed in sequential order without feedback loops. Every new phase is started only after the previous phase has been concluded. The complete product is not delivered until the final phase has been executed. Examples of this model are the Waterfall model and the Rapid Development Waterfall model. Linear models are suitable for routine and repetitive projects with few scope changes and clearly defined goals, solutions, and requirements. They lack, however, flexibility to respond to changing project goals and requirements.
- **Incremental** The incremental model resembles the linear model, but repeats three of the five phases to deliver parts of the end-product after each of the repetitions. The arrow leading back from the Deploy phase to the Build phase does not indicate client feedback, but a repetition of steps instead. Examples of this model are the staged delivery waterfall model or the feature-driven development model. Incremental models are used in similar conditions as linear models, but deliver business value earlier in the project life cycle, since useable parts of the end product are deployed throughout the project.
- **Iterative** As in the incremental model, the iterative model delivers partial product one after the other. However, the iterative model introduces means by which the client may review the increments and suggest changes in the build phase. Examples are the evolutionary development waterfall model, SCRUM, Rational Unified Process (RUP), and the Dynamic Systems Development Method (DSDM). These models require a more committed client than the models discussed above. In return, it provides a more flexible development approach to tackle projects in which the solution is not entirely known in advance.
- **Adaptive** When the solution is very little specified at the start of the project, an adaptive model may be used to learn from consecutive design iterations and converge to an acceptable end product. As in the iterative approach, clients provide feedback on partial releases of the end product. In response to this feedback, developers return to the build, or even design phase of the project. Examples of this type of model are the Adaptive Project Framework (APF) and Adaptive Software Development (ASD).
- **Extreme** As a last resort, when very little is known about the project goal and requirements, the extreme model may be adopted to provide maximal flexibility to the developer and its client. Models of this type, such as INSPIRE, use extremely short development cycles. Every cycle is reviewed by the client and this may result in build, design and even scope changes of the project. The extreme model is sometimes also referred to as "customer on-side" due to the intensive interaction with the client.

Agile development

Of the five software development models described above, the last three – iterative, adaptive, and extreme – are so-called agile development models (cf., Hazzan & Dubinsky, 2008). They have evolved in the last decades to deal with the growing complexity of most software development projects. High complexity calls for development models that effectively tackle uncertainty, i.e., models that allow for (Wysocki, 2006):

- Incomplete and possibly incorrect requirements
- Flexible, adaptive processes, fitted to the specific project
- Empowerment of development team members

- Fast response to project changes

Trends towards agile models are also found in the project management domain (cf., Wysocki, 2007). Generally speaking, agile models take a result-driven approach, as opposed to a plan-driven approach. As a result, agile models are better able to tackle more complex and more uncertain projects (where the project goals and possible solutions are not clear from the outset). Agile models see development projects as a discovery trip towards an acceptable solution. However, as discovery trips may fail, also agile development gives no guarantee for success. Also, it means a radical departure from existing business rules and the rules for project engagement. Therefore, switching to agile development requires careful preparations and guidance. For a more elaborate evaluation of agile software development, we refer to Abrahamsson et al. (2002).

Lessons from software engineering

In this section we have shown that game development projects show great resemblance with software engineering projects. We also demonstrated several software development models, among which three are agile models. Agile models are better able to deal with complex projects in which the goal and the requirements to the solution are not (well) known at the start.
Game design projects usually carry with them a lot of uncertainty around project goals and requirements to the solution. Game design is a multidisciplinary discipline which explains why it is difficult for a client to draw a complete picture of the project outcomes already at the start of the project. Additionally, many clients are not familiar with gaming as a method, which is another reason why requirements in and goals of gaming projects are at first instance often poorly described.
However, despite the complex and uncertain nature of many game design projects, the common development model as described by Duke and Geurts (see figure 1) is a linear model. From the discussion above it follows that although linear models may be appropriate for routine projects, they are not well suited to tackle complex and uncertain projects.
For the reasons mentioned above we believe that software engineering can teach the following important lessons to game designers:

1. Adapting the linear development model of Duke and Geurts to an agile model will lead to improved game design:
    a. Although they are no guarantee to success, agile models are better able to explore the client's needs and possible solutions (improved quality).
    b. Developers that employ an agile model spend less time on changing plans and more time on adding value to the project (improved efficiency).
2. Agile models, such as the iterative, adaptive, and extreme models require a much larger role for clients than linear models.

3 Towards agile game development

In recent years, many developers of computer games for entertainment purposes have moved towards agile models (Stacey and Nandhakumar, 2008). Inspired by this development, we extended the lessons learned from software engineering to a new approach to game design called Triadic Game Design (Harteveld, forthcoming; Harteveld et al., forthcoming). Triadic Game Design (TGD) is an attempt to formalize

the design process of games that simulate professional practices and policy evaluation. This approach, which emphasizes the multidisciplinary nature of game design projects, calls for an agile game development model. For this reason, we developed an agile game development model based on TGD and a workshop that can be used to initiate an agile game design process. Before we elaborate on this model and the workshop, we first briefly explain the philosophy behind TGD.

The TGD philosophy: Reality, Meaning and Play

According to the Oxford Dictionary a "triad" is "a group or set of three connected people or things". Based on the experience of designing a game to train levee inspectors called *Levee Patroller*, the TGD philosophy requires game designers to deal with one such triad (Harteveld, forthcoming; Harteveld et al., forthcoming). This triad consists of three components: Reality, Meaning, and Play. Below a short description is given of these components:

- **Reality** Games are connected to the real world. Reality could be represented by players (with their expertise and opinions) from the real world or by a representation of the real world inside the game. The Reality component determines the game subject, variables, and definitions. Criteria by which the game is evaluated include fidelity, realism, and validity. People affiliated with this world are the subject-matter experts.

- **Meaning** The Meaning component in TGD considers how a meaningful effect beyond the game experience can be intentionally achieved. It is related to various disciplines, such as learning sciences, psychology, and semiotics. It incorporates aspects as communication, learning, rhetoric, and opinions, while among its evaluation criteria we find reflection, transfer, and relevance. It is the world of teachers and storytellers.

- **Play** Aside from being related to a real world and having meaning, games are foremostly highly interactive and engaging tools that immerse people into a fictive situation. To achieve this, the Play component is concerned with selecting game elements, like actors, rules, resources, challenges, and competition to fulfil the evaluation criteria engagement, fun, and immersion. This component is affiliated with disciplines as media studies, game design, and human-computer interaction. This world belongs to the game designers and media specialists.

TGD asserts that a game needs to be developed equally along the three components described above. To achieve this, the designer needs to balance the three components. This is a difficult task, in particular because each of the components has its own evaluation criteria to determine the quality of the game and is related to different disciplines. The game design process can, therefore, be characterized as multidisciplinary. Such a nature asks for designers who are multiply talented or for collaboration with experts from different fields.

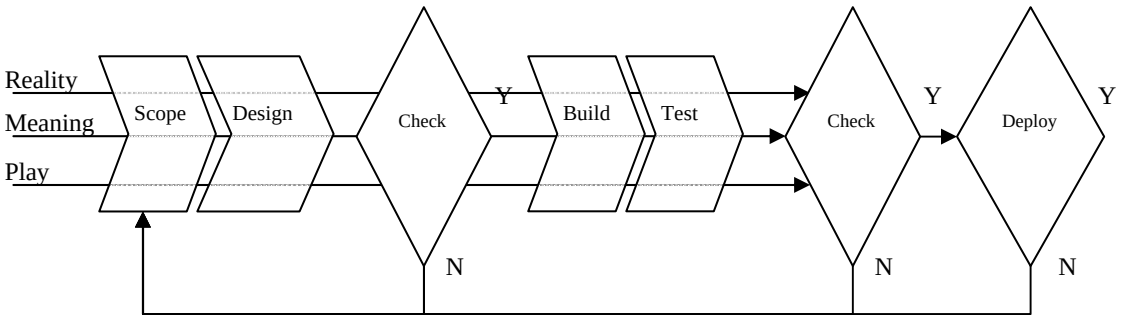
The TGD development model

We propose the TGD development model as is shown in Figure 4. To demonstrate the analogy to software development models, we use the software engineering phases in the figure. They can, however, be replaced by the game design phases from Figure 1.

The model prescribes that the three components Reality, Meaning, and Play are developed in parallel. Moreover, decision moments after the design and test phases and during the deploy phase are included. These two design choices are explained below.

First, we emphasise that games are systems that consist of many interrelated elements that jointly create an experience. Good games have a well-balanced structure: they are in harmony (Salen & Zimmerman, 2005). Such harmony is achieved, according to TGD, when each component is represented properly: only then, players go through a meaningful experience, related to reality, in a playful manner. Balancing a game calls for managing various *trade-offs* since optimising the game from the perspective of one component (e.g., Reality) may compromise the quality from the perspective of another component (e.g., Play). If disagreement exists on a design issue when viewed from different perspectives the designers face a dilemma. To detect dilemmas in an early stage and to resolve them properly, designers need to consider as many components at the same time as possible. Therefore, a parallel development process along the three components Reality, Meaning, and Play is recommended in our development model.

Figure 4 The Triadic Game Design development model



Secondly, the TGD development model was developed to tackle complex game design projects of which the goals and product requirements are often not known from the start. As was shown above, the extreme software development model was devised for exactly these purposes. Therefore, we based the TGD development model on the extreme model. As in extreme development, multiple decision moments exist in TGD at which the designer and client are able to apply changes to the built artefact, the design, and even scope definition. Moreover, subsequent development iterations are short, allowing rapid exploration of the design requirements and project goals.

The TGD development model differs from the "standard" game design approach proposed by Duke and Geurts: it does not prescribe a "stepwise, waterfall procedure". It does not tell designers that they need to finalize one step before moving to the next to create a game. Instead, it emphasizes that game development is a parallel and iterative process:

- Designers need to take the worlds of Reality, Meaning and Play into account at the same time to balance the game;
- No complete scope definition is required at the start of the project. Instead, the scope definition is an evolving document;

- TGD develops games with "the customer on-side", allowing for collaborative adaptation of the scope, design, and built artefact.

A TGD workshop

To implement the first iteration cycle of the TDG development model shown above, we developed a TGD workshop (Harteveld & Van den Bergh, 2008). The workshop brings together the client and experts that fulfil the roles attached to the different TGD components (subject-matter experts, teachers, and game designers). The aim is to facilitate game development and to provide all parties with a common vocabulary and understanding of game design.
During the workshop[1], participants are introduced to the components Reality, Meaning, and Play consecutively. With each acquaintance participants are asked to work in groups on an assignment. The first assignment involves creating a model of real-world system on which the game focuses. Examples are environmental policies or business processes. In the second assignment, related to the Meaning component, participants determine the function of their game. A function can be to design a game for raising awareness or for training physical skills. Finally, the participants select a game type (e.g., a management game or board game) to agree on the Play component. After the three assignments they are required to design a complete game design concept. With each assignment groups are presented *worksheets* that help to give them an idea of what they need to consider. For example, the Play worksheet indicates that groups should consider what the goals are *in* the game.

At the end of the workshop, each group presents its draft game concept. The participants are informed that this concept is in fact the result of the first iteration in the TGD development model. The workshop therefore serves two purposes: to start up the game development process and to align the parties involved in the design.

4 Conclusions

Many game design projects do not meet the cost, time, and quality objectives that were promised to the client. Commonly reported reasons are the technical complexity, poor management, unrealistic expectations, and incomplete and vague goals and requirements of these projects. There is a clear need for a better approach to performing game design projects. This may be accomplished by more stringent use of existing development models, or by the development of new models. In this paper we have studied the second possibility and proposed an agile model to develop games that simulate professional practice and policy evaluation.

We were inspired by a related design discipline, software engineering, that has faced similar problems due to the ever growing complexity of software development projects. In response to this, new software development models have evolved during the past decades. These so-called agile models provide greater flexibility to deal with uncertain project goals and product requirements.

[1] The workshop is very flexible. Various variations exist, in one of which the workshop itself is designed as a game. So far, the workshop has been applied 12 times. Most of its applications have been purely educational.

In this paper we focused on games that simulate professional practice and policy evaluation. From the analysis we conclude that the most widely used development model for this type of games is of the linear type. Although linear models are suitable for conducting routine and relatively simple game design projects, they are less well equipped to tackle large and complex projects. In contrast, in the computer entertainment gaming industry we found that developers are slowly moving towards agile models to improve project performance.

Moreover, it is concluded that software engineering may teach game designers two important lessons. Game developers should:

1. apply agile development models to improve the quality and efficiency of large, complex game design projects;
2. strengthen the interaction with their clients in order to enable agile development.

We applied these findings to yield a promising first step towards a possible agile game development model. This model is inspired by the *extreme* software development model, which is ultimately equipped to improve complex software development projects, and by a formal design approach called Triadic Game Design. Our development model prescribes the simultaneous development of three game components – Reality, Meaning, and Play – and an iterative development of project scope, design, and built artefact. For a practical implementation of our approach, we devised and successfully conducted a workshop that can be used to initiate an agile game development project.

Future work in this line of research includes evaluating the performance of our agile game development approach when compared to more traditional approaches. Also, we aim to improve the TGD workshop, by implementing a parallel process of developing Reality, Meaning, and Play and by extending the workshop to include several iteration cycles and criteria for convergence. Finally, we look at incorporating lessons from collaboration engineering (cf., Briggs et al., 2003; Kolfschoten & De Vreede, 2009) to improve agile game design.

References

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile Software Development Methods: Review and Analysis*. VTT Publications 478.

Briggs, R.O., G-J. de Vreede, & J.F. Nunamaker Jr (2003) *Collaboration engineering with Thinklets to pursue sustained success with group support systems*. Journal of management information systems, 19(4) pp. 31- 63.

Duke, R.D. (1974). *Gaming: the futures language.* Beverly Hills: sage Publications.

Duke, R.D. & Jac L.A. Geurts (2004) *Policy games for strategic management*: *Pathways into the Unknown*. Amsterdam: Dutch University Press.

Harteveld, C. (forthcoming). Triadic game design: balancing reality, meaning and play.

Harteveld, C., & Van den Bergh, R. (2008). *Serious game design workshop*. In P. Markopoulos, J. Hoonhout, I. Soute, & J. Read (Eds.), *International Conference*

on Fun and Games, Eindhoven, the Netherlands, October 20-21 2008 (pp. 68-73). Eindhoven, the Netherlands: Eindhoven University of Technology.

Harteveld, C., R. Guimaraes, I.S. Mayer, & R. Bidarra (forthcoming). *Balancing reality, meaning and play: the design philosophy of LEVEE PATROLLER*. Simulation & Gaming.

Hazzan, O. &Y. Dubinsky (2008) *Introduction to agile software development*. In: Agile software engineering. Dubinsky, Y. and O. Hazzan (Eds.). London: Springer-Verlag.

Kolfschoten, G.L. & G-J. de Vreede (2009) *A design approach for collaboration processes: a multimethod design science study in collaboration engineering*. Journal of management information systems, 26(1) pp. 231- 262.

McHaney, R., White, D., and Heilman, G.E. (2002) *Simulation Project Success and Failure: Survey findings*. Simulation & Gaming, Vol. 33, No. 1, pp. 49-66.

Pfleeger, S.L. & J.M. Atlee (2005) *Software engineering : theory and practice*, 3rd ed. Upper Saddle River: Prentice-Hall.

Salen, K. & Zimmerman, E. (2004). *Rules of play: game design fundamentals*. Cambridge, MA: MIT Press.

Stacey, P. and Nandhakumar, J. (2008) Opening up to agile games development. Communications of the ACM, Vol. 51, No. 12, pp. 143 – 146.

Wysocki, R.K. (2006) *Effective software project management.* Indianapolis: Wiley.

Wysocki, R.K. (2007) *Effective Project Management: Traditional, Adaptive, Extreme*. 4th Edition. Indianapolis, IN: John Wiley & Sons, Inc.