Methodology

Simulation

Simulation: Transactions of the Society for Modeling and Simulation International
2015, Vol. 91(4) 307–319

DOI: 10.1177/0037549715572673
sim.sagepub.com
SAGE SCS

GAMED: digital educational game development methodology

Serdar Aslan and Osman Balci*

Abstract

Development of a game in the form of software for game-based learning poses significant technical challenges for educators, researchers, game designers, and software engineers. The game development consists of a set of complex processes requiring multi-faceted knowledge in multiple disciplines such as digital graphic design, education, gaming, instructional design, modeling and simulation, psychology, software engineering, visual arts, and the learning subject area. Planning and managing such a complex multidisciplinary development project require a unifying methodology and should not be performed in an ad hoc manner. This paper presents such a methodology named GAMED (diGital educAtional gaMe dEvelopment methoDology). GAMED consists of a body of methods, rules, and postulates and is embedded within a digital educational game life cycle. The life cycle describes a framework for organization of the phases, processes, work products, quality assurance activities, and project management activities required to develop, use, maintain, and evolve a digital educational game from birth to retirement. GAMED provides a modular structured approach for overcoming the development complexity and guides the developers throughout the entire life cycle.

Keywords

Digital game development, educational game development, game development life cycle, game development methodology, game development processes, game development work products, game development workflows, game quality assurance, game-based learning

1. Introduction

A *Digital Educational Game* (DEG) is a game created for the purpose of teaching a subject in the form of software that runs on a computer such as desktop, laptop, handheld, or game console.

Johnson et al.[1] identify game-based learning as a technology expected to gain widespread usage. Gee[2] indicates that the theory of learning in good DEGs fits better with the modern, high-tech, global world today's children and teenagers live in. Prensky[3] points out that DEG-based learning (a) meets the needs and learning styles of today's and the future's generations of learners, (b) is motivating, because it is fun, (c) is enormously versatile, adaptable to almost any subject, information, or skill to be learned, and (d) when used correctly, is extremely effective.

Felicia[4] identifies one of the DEG-based learning challenges as ''More developers need to be informed of best practices pertaining to the design of successful educational games.'' Development of a DEG in the form of software for DEG-based learning poses significant technical challenges for educators, researchers, game designers, and software engineers. The DEG development consists of a set of complex processes requiring multi-faceted knowledge in multiple disciplines such as digital graphic design, education, gaming, instructional design, modeling and simulation, psychology, software engineering, visual arts, and the learning subject area. Planning and managing such a complex multidisciplinary development project require a unifying methodology and should not be performed in an ad hoc manner.

The purpose of this paper is to present such a methodology named GAMED (diGital educAtional gaMe dEvelopment methoDology). GAMED consists of a body of methods, rules, and postulates and is embedded within a DEG life cycle consisting of four phases and a dozen processes.

Department of Computer Science, Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, VA, USA

*SCS member.

Corresponding author:
Osman Balci, Department of Computer Science, MC 0106, Virginia Tech, Blacksburg, VA, USA.
Email: balci@vt.edu

This paper is organized as follows. After an introduction in Section 1, Section 2 advocates the motto ''Quality is Job 1!'' and presents a DEG quality assurance (QA) strategy. Section 3 describes the DEG life cycle, which provides the foundation of GAMED. Concluding remarks are given in Section 4.

2. Digital educational game quality assurance

As the saying goes ''Quality is Job 1!'' The ultimate goal of a DEG development project is to produce a game in the form of a software application with desirable quality characteristics such as challengeability, effectiveness, engageability, enjoyability, interactivity, transformativeness, and usability.

DEG QA refers to the planned and systematic activities that are established throughout the DEG life cycle to substantiate adequate confidence that the DEG software application possesses certain characteristics required for a set of intended uses of the DEG.

The more comprehensive and detailed the overall game quality evaluation is, the more confidently a decision can be reached for the acceptability of the game. Four major perspectives or four Ps influence the game quality as depicted in Figure 1.

DEG quality evaluation can be approached from any one of the four Ps, but a combination of all four will provide the best balance and result in a much higher level of confidence in the game quality. Therefore, we advocate the strategy outlined below.

The DEG QA strategy should involve the measurement and integrated evaluation of the following in a particular life cycle stage (e.g. game design stage):

- output work *Product* (or artifact), (e.g. game design specification);
- *Process* used in creating the output work product, (e.g. game design);
- quality of the *People* employed in executing the process to create the work product; and
- *Project* characteristics (e.g. configuration management, risk management, project planning, project monitoring and control).

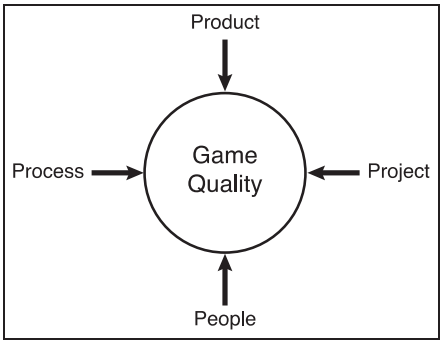

Figure 1. Four Ps influencing the game quality.

DEG quality is the degree to which the DEG possesses a desired set of characteristics. An example set of quality indicators is provided below for evaluating the quality of a DEG.

1. *Acceptability*: the degree to which the DEG fulfills its requirements and learning objectives.
2. *Challengeability*: the degree to which the user finds the DEG to be exciting, stimulating, and inspiring to play.
3. *Clarity*: the degree to which the DEG is unambiguous and understandable.
4. *Effectiveness*: the degree to which the DEG improves the effectiveness of learning the subject in a significantly better way in comparison to other pedagogies.
5. *Engageability*: the degree to which the user is captivated and addicted by DEG playing.
6. *Enjoyability*: the degree to which the user finds the DEG playing to be fun.
7. *Interactivity*: the degree to which the user actively interacts with the DEG during playing.
8. *Localizability*: the degree to which the DEG can easily be adopted, preferably via preferences or options (a) to satisfy the needs of languages other than English, and (b) to local standards such as decimal separator, currency symbol, time zone, calendar, etc.
9. *Rewardability*: the degree to which the DEG gives rewards (e.g. points, money, trophies, certificates) to the user so that the user feels a sense of accomplishment.
10. *Simplicity*: the degree to which the DEG can be understood without difficulty.
11. *Transformativeness*: the degree to which the DEG transforms the subject learning in a significantly better way in comparison to other pedagogies.
12. *Usability*: the degree to which the DEG can easily be employed for its intended use.

3. Digital educational game life cycle

GAMED requires the employment of the DEG life cycle presented in Figure 2 and is embedded within that life cycle. The DEG life cycle provides the foundation of GAMED.

The DEG life cycle represents a *framework* for organization of the phases, processes, work products, quality

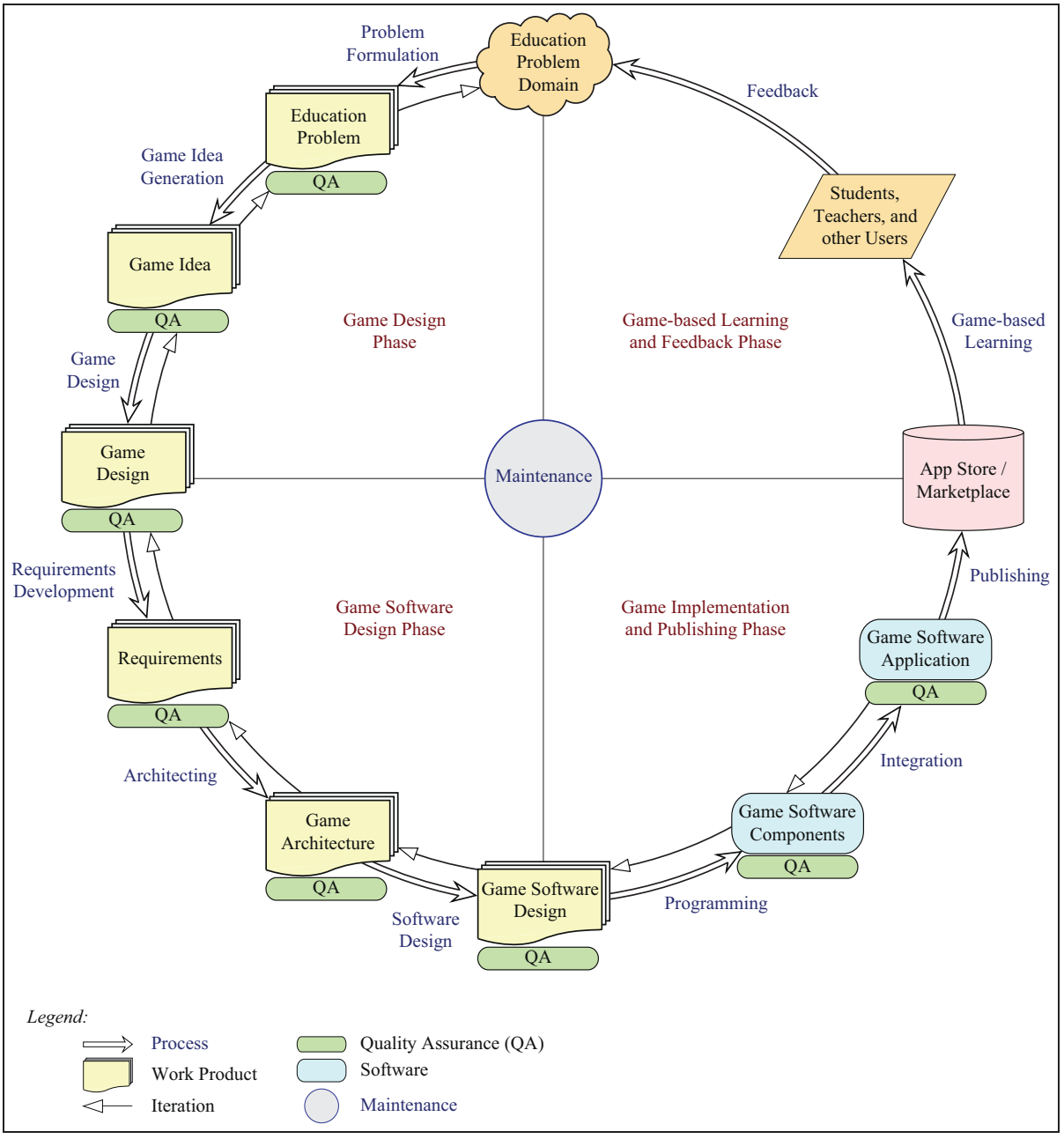

Figure 2. Digital educational game life cycle.

assurance activities, and project management activities required to develop, use, maintain, and evolve a DEG from birth to retirement. The DEG life cycle modularizes and structures the development of a DEG and provides guidance to game designers, software engineers, and project managers.

The DEG life cycle consists of four phases: game design phase, game software design phase, game implementation and publishing phase, and game-based learning and feedback phase. Each phase consists of a number of stages. Stages are named after the processes, e.g. game idea generation stage, architecting stage, programming stage. Each stage consists of a process and a work product, e.g. requirements development stage consists of the requirements development process producing the requirements specification document as a work product.

The DEG life cycle enables DEG development to be viewed from four perspectives (or Ps): Process, Product, People, and Project. The DEG life cycle (a) specifies which work *Product* to produce by executing which *Process* together with the integrated quality assurance activities, (b) provides valuable guidance for *Project*

management, and (c) identifies areas of expertise in which to employ qualified *People*.[5]

The DEG life cycle represents the processes in a logical order starting with ''Education Problem Domain'' using double-line arrows as depicted in Figure 2. Although the arrows show a sequential progress, the life cycle should *not* be interpreted as strictly sequential. The sequential representation of the double-line arrows is intended to show the direction of workflow throughout the life cycle.

The DEG life cycle is *iterative* in nature and reverse transitions are expected. The backward solid line arrows between work products represent the iterative nature of the development. For example, a problem identified in game software design may require changes in game design and hence, we bounce back to an earlier process and redo the work. We typically bounce back and forth between the processes until we achieve sufficient confidence in the quality of the work products throughout the life cycle.[5]

A DEG life cycle process, represented by a double-line arrow, is executed to produce a work product. For example, the *game design* process is executed to produce the work product *game design specification*, the *architecting* process is executed to produce the work product *game architecture specification*, and the *programming* process is executed to produce the work product *game software components*.

QA shown in rounded rectangle symbols in Figure 2, cannot be designated as a stage in the DEG life cycle. QA consists of continuous activities conducted hand in hand as integrated with every DEG life cycle task.[6] We represent this principle by placing the QA symbol below each work product as depicted in Figure 2.

3.1. Problem formulation

Education Problem Domain is the starting point for the DEG life cycle as depicted in Figure 2. A DEG is developed to address a specific problem formulated in this domain. The process of *Problem Formulation* can be executed by following the steps described below.

Step 1. *Identification of the education problem*. Identify a subject that poses serious challenges for learning, e.g. learning fractions by middle school students poses serious challenges.

Step 2. *Significance of the education problem*. Explain why the identified education problem is critically important to solve.

Step 3. *Identification of the state of the art in solving the education problem*. Identify the state-of-the-art pedagogies under which the education problem is currently being solved. Explain why the existing pedagogies are not good enough.

Step 4. *Effectiveness of game-based learning for solving the education problem*. Describe why game-based learning would provide a more effective approach to solving the education problem in comparison with the existing pedagogies. *Continue with the DEG life cycle only if game-based learning is found to be an effective approach to solving the education problem*.

Step 5. *Definition of the education problem*.

1. Establish the problem domain boundary. A *problem domain boundary* identifies which elements will be included and which elements will be excluded. What to include and what not to include is considered as the art of balancing the opposites. The boundary must be large enough to encapsulate all essential elements of the problem domain.
2. Gather data and information about the problem domain within the established boundary.
3. Identify the stakeholders and decision makers who would be interested in the solution of the education problem.
4. Specify the needs and objectives of the stakeholders and decision makers identified.
5. Identify and specify the constraints.
6. Specify all assumptions made clearly and explicitly and validate each assumption individually.
7. Produce a well-structured definition of the education problem.

Step 6. *Specification of the game-based learning objectives*. Acceptability of the DEG to be developed will be judged with respect to the learning objectives, which must be clearly and explicitly stated.

As the saying goes, ''A problem correctly formulated is half solved.'' Sufficient time and effort must be expended at this life cycle stage to make sure that Type III Error is not committed. (Type III Error is the error of solving the wrong problem.[7])

Execution of the *Problem Formulation* process produces the *Education Problem Specification* document as a work product containing the results of performing the steps described above.

The QA strategy for the problem formulation stage involves the measurement and integrated evaluation of the (a) education problem specification document as a work *Product*, (b) problem formulation *Process*, (c) quality of the *People* employed in executing the problem formulation process, and (d) *Project* characteristics that are specific to this life cycle stage.

Table 1. Game idea generation process.

SG 1	Perform quality assurance continuously
	SP 1.1 Employ the four Ps for game quality assurance
	SP 1.2 Employ game quality indicators
SG 2	Assemble game idea generation team
	SP 2.1 Identify a moderator
	SP 2.2 Identify a scribe
	SP 2.3 Identify teachers
	SP 2.4 Identify game players
	SP 2.5 Identify game developers
	SP 2.6 Identify a quality bearer
SG 3	Stimulate creativity
	SP 3.1 Select an acceptable game genre
	SP 3.2 Combine different game genres
	SP 3.3 Generate ideas from other media
	SP 3.4 Generate ideas from real life
	SP 3.5 Generate ideas from imagination
SG 4	Hold game idea generation meetings
	SP 4.1 Provide information to team members
	SP 4.2 Task team members to prepare
	SP 4.3 Execute game idea generation meeting
	SP 4.4 Document each game idea generated
	SP 4.5 Produce and distribute meeting notes
SG 5	Process generated game ideas
	SP 5.1 Group generated game ideas
	SP 5.2 Perform comparative evaluation
	SP 5.3 Select the best game idea
SG 6	Produce game idea specification document
	SP 6.1 Define learning objectives and target audience
	SP 6.2 Describe game story
	SP 6.3 Identify game characters
	SP 6.4 Describe gameplay
	SP 6.5 Describe game levels
	SP 6.6 Describe quality assurance results

3.2. Game idea generation

We describe the game idea generation process based on a structure advocated by the Software Engineering Institute (SEI)[8] in terms of Specific Goals (SGs) and their associated Specific Practices (SPs) as shown in Table 1.

SG 1 Perform quality assurance continuously

QA is not a stage but a continuous set of activities conducted hand in hand together with the development activities. The QA strategy described in Section 2 should be followed. To achieve this goal, the following specific practices are employed.

SP 1.1 Employ the four Ps for game quality assurance: The QA strategy for the game idea generation stage of the life cycle involves the measurement and integrated evaluation of the (a) game idea specification document as a work *Product*, (b) game idea generation *Process*, (c) quality of the *People* employed in executing the game idea generation process, and (d) *Project* characteristics that are specific to this life cycle stage.

SP 1.2 Employ game quality indicators: Game idea is evaluated by employing game quality indicators such as the ones presented in Section 2.

SG 2 Assemble game idea generation team

The game idea generation process should be executed in a team environment. The team holds as many game idea generation meetings as needed. The team may include subject matter experts (e.g. teachers), game-based learning experts, game designers, game players (e.g. students), and software engineers. To achieve this goal, the following specific practices are employed.

SP 2.1 Identify a moderator: The moderator presides over the game idea generation meetings to facilitate the discussions and encourage team members to be creative.

SP 2.2 Identify a scribe: The scribe records the participants' comments, game ideas proposed, decisions reached, action items, and produces meeting notes after the meeting.

SP 2.3 Identify teachers: Teachers provide expertise about the state of the art in pedagogies for teaching the subject, define the learning objectives, and help the team understand the game-based learning subject.

SP 2.4 Identify game players: Experienced entertainment or educational game players can generate game ideas based on their playing experience.

SP 2.5 Identify game developers: Game developers provide expertise about game design, software engineering, software graphics and visualization, game development tools and techniques, and game deployment platforms such as mobile devices.

SP 2.6 Identify a quality bearer: The quality bearer provides feedback to the team about whether a proposed game idea can lead to the development of a game with acceptable quality.

SG 3 Stimulate creativity

Creativity refers to the ability to produce a new game idea through imaginative skill, richness of ideas, and originality of thinking. To achieve this goal, the following specific practices are employed.

SP 3.1 Select an acceptable game genre: Creativity can be stimulated by narrowing the game idea generation under a specific game genre such as adventure game, multiplayer online game, puzzle game, or role-playing game.

SP 3.2 Combine different game genres: Combining the features of different types of games may lead to the creation of a new game type with innovative composite characteristics.

SP 3.3 Generate ideas from other media: Identifying stories in media such as books, magazines, movies, newspapers, and television can boost creativity for coming up with a game idea.

SP 3.4 Generate ideas from real life: Real-life situations can be the basis of a game idea such as the Farmville game,[9] which simulates the activities of a farm.

SP 3.5 Generate ideas from imagination: Imagination is considered to be the foundation for a creative mind. Albert Einstein said, ''Imagination is more important than knowledge. For knowledge is limited to all we now know and understand, while imagination embraces the entire world, and all there ever will be to know and understand.''

SG 4 Hold game idea generation meetings

Creative team brainstorming should take center stage for a game idea generation meeting. It encourages collaboration, enables team members to feel more open to bouncing ideas off one another, facilitates different perspectives, and improves the team's ability to think outside the box. To achieve this goal, the following specific practices are employed.

SP 4.1 Provide information to team members: All team members receive the *Education Problem Specification* document and any other related information before the meeting.

SP 4.2 Task team members to prepare: Adequate time is given to the team members to prepare themselves for the meeting by studying the information provided and conducting a literature review to familiarize themselves with the education problem domain.

SP 4.3 Execute game idea generation meeting: The meeting is executed based on the roles and responsibilities of the team members specified under SG 2.

SP 4.4 Document each game idea generated: All game ideas generated during the meeting are properly documented.

SP 4.5 Produce and distribute meeting notes: The scribe produces the meeting notes and distributes them to the team members after the meeting.

SG 5 Process-generated game ideas

After a game idea generation meeting, generated game ideas are analyzed, grouped, and compared with each other. If a game idea with sufficient quality cannot be selected, the SPs of SG 4 above should be performed again. The SPs of SG 4 and SG 5 are performed in an iterative manner, bouncing back and forth, until a game idea with sufficient quality is selected. To achieve this goal, the following specific practices are employed.

SP 5.1 Group generated game ideas: Some game ideas may be related to each other and grouping them facilitates the analysis of their characteristics. Combining the features of related game ideas can produce a better composite game idea.

SP 5.2 Perform comparative evaluation: Identified game ideas in different groups are comparatively evaluated using quality indicators such as the ones listed in Section 2.

SP 5.3 Select the best game idea: The game idea with the best quality characteristics is selected. The SPs of SG 4 are executed again if no selection can be made.

SG 6 Produce game idea specification document

Execution of the *Game Idea Generation* process produces the *Game Idea Specification* document as a work product, which describes at least the following: learning objectives, target audience, game story, game characters, gameplay, game levels, and quality assurance results. To achieve this goal, the following specific practices are employed.

SP 6.1 Define learning objectives and target audience: The game-based learning objectives for a particular subject and the intended target audience (e.g. middle school students) are clearly and explicitly defined.

SP 6.2 Describe game story: A high-level description of the game story is provided.

SP 6.3 Identify game characters: Characters significantly influence game quality. A good character is someone a player can relate to. Game characters are explicitly identified.

SP 6.4 Describe gameplay: Gameplay defines how players interact with a game under a set of rules and mechanics, as distinct from the game's graphics and sound effects. A high-level description of the gameplay is provided.

SP 6.5 Describe game levels: Games may be decomposed into modules (levels) to overcome complexity. The game levels are defined in increasing difficulty requiring the player to complete a lower level to advance to the next level.

SP 6.6 Describe quality assurance results: The results of applying the QA strategy described in Section 2 and under SP 1.1 are fully documented.

3.3. Game design

The game design process takes the *Game Idea Specification* document as input and produces the *Game*

Design Specification document as a work product. We advocate the use of the *Spiral Game Design* strategy as depicted in Figure 3.[10]

The Spiral Game Design strategy employs the following principles.

1. *Evolutionary development*: Develop the game design gradually in increasing level of detail as more experience and insight are obtained through prototyping and playtesting.
2. *Incremental development*: Develop the game design in increments by identifying each increment with a version number and with a set of design features under a configuration baseline.
3. *Iterative refinement*: Refine the level of game design granularity in an iterative manner based on experience and insight obtained through prototyping and playtesting.
4. *Progressive elaboration*: Add details to the game design progressively by elaborating (i.e. expanding and embellishing) the game design characteristics based on experience and insight that become available through prototyping and playtesting.

The game design process consists of four major activities: prototyping, playtesting, evaluation, and risk analysis.

The spiraling depicted in Figure 3 starts with the Game Idea Specification document, based on which the *Prototyping* activity is conducted to produce Game Design Prototype 1. In the next activity *Playtesting*, potential players of the game experiment with the prototype under simulated conditions for the purpose of discovering design flaws, providing insight, and giving feedback. The *Evaluation* activity is conducted to judge the game quality using quality indicators such as the ones presented in Section 2. The *Risk Analysis* activity is conducted to identify potential risks and determine how to mitigate them. Based on the experience and insight gained from playtesting, evaluation, and risk analysis, an improved Game Design Prototype 2 is created. The spiraling, consisting of prototyping–playtesting–evaluation–risk analysis, is repeated to develop Game Design Prototype 3. The spiraling continues until the game design is judged to possess acceptable quality in the *N*th spiraling. The number *N* changes from one game design to another depending on the size and complexity of the game design. The Game Design Prototype *N* is described in the *Game Design Specification* document as the work product produced by the game design process.

The activities of the game design process are described below.

3.3.1 Prototyping. Prototyping is conducted to develop a prototype of a game design to enable potential players of the game experiment with the expected game functionality under simulated conditions for the purpose of discovering

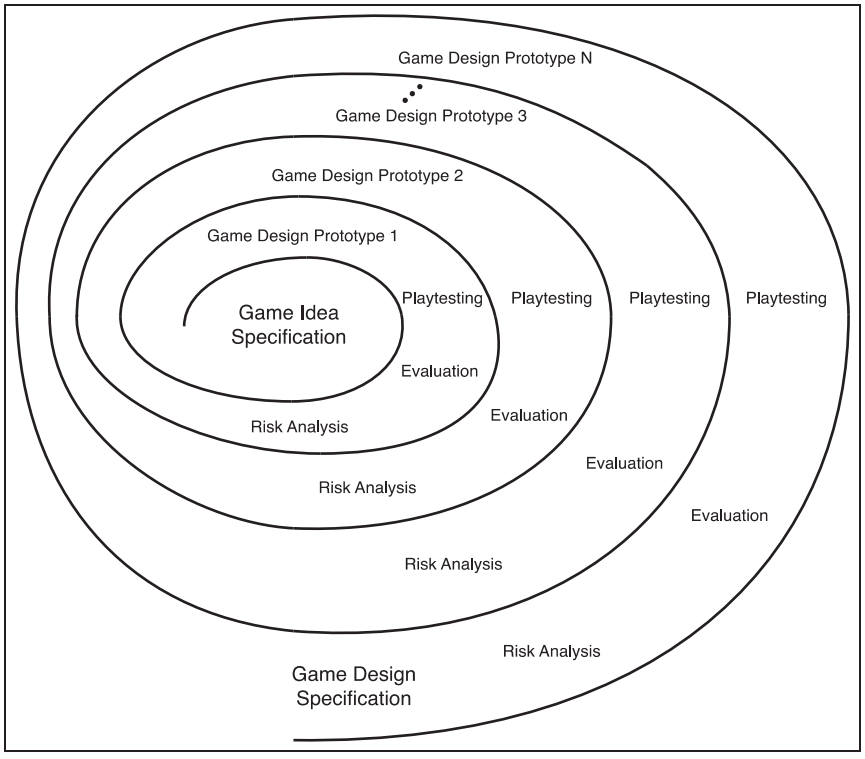

Figure 3. Spiral game design.

design flaws, providing insight, and giving feedback. A game design prototype (a) represents the game's user interface to enable a player to interact with the game in a simulated manner, (b) is built rapidly and cheaply, (c) must be easily modifiable, and (d) facilitates communication between the game designers and potential players.

Two major types of game design prototyping exist.

1. *Paper Prototyping*: Used to build the game design prototype in the form of paper representing the game's graphical user interface and the player's interactions with the game using hand-sketched or computer-printed images.
2. *Software Prototyping*: Used to build the game design prototype in the form of interactive software representing the game's graphical user interface and the player's interactions with the game. A software tool can be used to rapidly put together the game's user interface graphics using images downloaded from the web.

For large and complex game designs, prototyping can start with paper prototyping in the early stage of the game design and later continues with software prototyping.

3.3.2 Playtesting. *Playtesting* refers to the activity in which potential users experiment with a prototype of a game design under development for the purpose of discovering design flaws, providing insight, and giving feedback. The prototype can be a paper or software prototype.

Playtesting is conducted under the following three steps.

Step 1. *Preparation for playtesting*: This step consists of the following tasks.
   1. *Define goals for playtesting*: Playtesting is conducted in a goal-directed manner. All goals should be clearly and explicitly specified.
   2. *Select a playtesting method*: A playtesting method such as group testing, individual testing, interviews, feedback forums, guided, or unguided is selected.
   3. *Develop a procedure for playtesting*: A playtesting procedure is developed to describe how the playtesting will be performed step by step and what test data and simulated conditions will be employed.
   4. *Recruit people for playtesting*: Qualified people with multifaceted knowledge are recruited to perform playtesting.

Step 2. *Execution of playtesting*: Playtesting is executed according to the steps of the playtesting procedure in a controlled environment where players are observed while playing the game and observations and notes are recorded concurrently.

Step 3. *Documenting playtesting results*: Recorded videos of playtesting, observations collected, notes taken, and any other recordings are all documented.

3.3.3 Evaluation. The Evaluation activity is conducted by analyzing the documented playtesting results and judging the game design quality. Each of the game quality indicators such as the ones given in Section 2 is evaluated based on the documented playtesting results. The evaluation aims to identify game design flaws. Based on the evaluation results, the game design is improved and a new game design prototype is created to reflect the improvements.

3.3.4 Risk analysis. A game development project management is expected to include proactive risk management, which consists of risk identification, risk analysis, risk planning, and risk monitoring. Risks, including the ones described below, should be (a) identified, (b) prioritized in terms of their estimated probability of occurrence and consequences, (c) monitored during the project, and (d) mitigated under a plan.[6,11]

1. *Product risk*: the probability that the game's characteristics will be unsatisfactory.
   a) *Acceptance risk*: the probability that the game will not possess sufficient *Acceptability*, the degree to which the game fulfills its requirements and learning objectives.
   b) *Integration risk*: the probability that the game modules will not be successfully integrated.
   c) *Performance risk*: the probability that the game will be too slow to play.
   d) *Usability risk*: the probability that the game will not be easily employed for its intended use.
   e) *Supportability risk*: the probability that the game cannot be properly maintained after its delivery.
   f) *Utility risk*: the probability that the game will be less useful than required by its intended users.
2. *Resource risk*: the probability that the game development will exceed the allocated resources such as budget and time.
   a) *Cost risk*: the probability that the game development will exceed the budgeted amount.
   b) *Schedule risk*: the probability that the game will not be delivered by the required deadline.

3. *Technology risk*: the probability that the software and hardware technologies used in developing the game will be unsatisfactory.
4. *People risk*: the probability that some adverse circumstances will occur due to the game development team members.

3.4. Requirements development

Requirements development is the process of identification and specification of functional and non-functional requirements for the DEG and game software application. This process takes the game design specification document as input and produces the requirements specification document as an output work product.

Functional requirements are identified by employing use cases created based on the game design specification document. A *use case* describes a particular interaction between the player and the game or between two game software components. Use cases enable us to decompose the complex game software functionality into small pieces of functionality, for which functional requirements can be identified in a more effective manner. Use cases turn themselves into classes in an object-oriented software design and significantly facilitate the transition from requirements to software design. Use cases also enable life cycle traceability between a requirement and a part of executable code.

Non-functional requirements are identified based on the game design, the education problem specification, and the game sponsor's objectives. A non-functional requirement specifies a criterion applicable for the entire game such as (a) running on all iOS mobile devices, (b) playable only in landscape device orientation, (c) supporting all screen resolutions, and (d) requiring internet connection.

A requirement is specified using ''shall'' in a sentence or a paragraph. For example, ''The player *shall* slice the candy bar into 2 to 10 equal pieces by using a swipe gesture from top to bottom over the candy bar.'' However, the requirement specification should not be viewed as a sentence or a paragraph. It must be engineered as a product required to possess certain quality characteristics such as the ones given below.

1. *Requirements accuracy* is the degree to which the requirements possess sufficient transformational and representational correctness.
   a) *Requirements verity* is evaluated by conducting requirements verification. *Requirements verification* is substantiating that the requirements are transformed from higher levels of abstraction into their current form with sufficient accuracy. Requirements verification addresses the question of ''Are we creating the requirements right?''
   b) *Requirements validity* is evaluated by conducting requirements validation. *Requirements validation* is substantiating that the requirements represent the *real* needs of the player (student) and the education problem domain with sufficient accuracy. Requirements validation addresses the question of ''Are we creating the right requirements?''
2. *Requirements clarity* is the degree to which the requirements are unambiguous and understandable.
   a) *Requirements unambiguity* is the degree to which each statement of the requirements can only be interpreted one way.
   b) *Requirements understandability* is the degree to which the meaning of each statement of the requirements is easily comprehended by all of its readers.
3. *Requirements completeness* is the degree to which all parts of a requirement are specified with no missing information, i.e. each requirement is self-contained.
4. *Requirements consistency* is the degree to which the requirements are specified using uniform notation, terminology, and symbology, and any one requirement does not conflict with any other.
5. *Requirements feasibility* is the degree of difficulty of implementing a single requirement, and simultaneously meeting competing requirements. Sometimes requirements conflict with each other. It may be possible to achieve a requirement by itself, but it may not be possible to achieve a number of them simultaneously.
6. *Requirements modifiability* is the degree to which the requirements can easily be changed.
7. *Requirements stability* is the degree to which the requirements are changing while the game software application is under development, and the possible effects of the changing requirements on the project schedule, cost, risk, quality, functionality, design, integration, and testing of the application.
8. *Requirements testability* is the degree to which the requirements can easily be tested. A testable requirement is the one that is specified in such a way that pass/fail or evaluation criteria can be derived from its specification.
9. *Requirements traceability* is the degree to which the requirements related to a particular requirement can easily be found.

3.5. Architecting

Architecting is the process of creating and specifying an architecture for a network-centric DEG software application. This process is required only for those DEGs

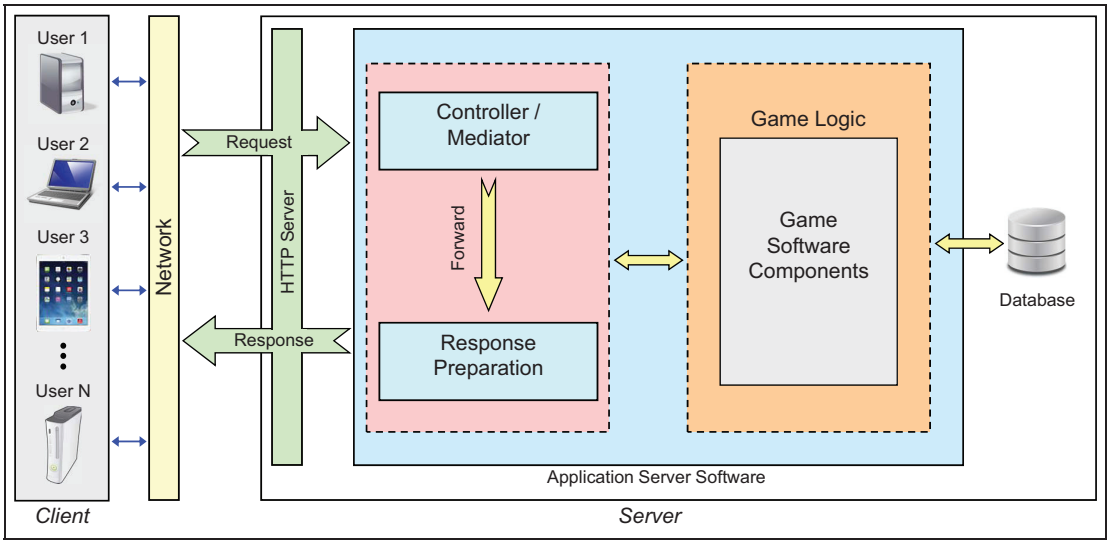

Figure 4. Network-centric game client-server architecture.

requiring network connection. It can be skipped for standalone DEGs.

The process of architecting takes the requirements specification document as input and produces a game architecture specification document as the output work product. It can be conducted under the guidelines provided by Chigani and Balci.[12]

The *game architecture specification* refers to the fundamental organization of DEG software application components that interoperate over a network (e.g. internet), relationships among the DEG components, and the principles and guidelines governing the design and evolution of those components. DEG architecture is specified by typically using DoD Architecture Framework (DoDAF), which provides 52 models (diagrams) for representing an architecture.[13–15]

Commonly used architectures include client-server architecture, service-oriented architecture, and peer-to-peer architecture. The architecting process can be conducted by (a) selecting a known architecture, (b) composing an architecture from a set of known architectures, or (c) creating a new architecture.

Figure 4 illustrates a network-centric game client-server architecture. The game consists of two parts: the part that runs on a client computer (e.g. desktop, laptop, handheld, or game console) and the part that runs on a server computer. These parts interoperate with each other over a network under the client-server architecture layers shown in Figure 4.

Massively Multiplayer Online Game is a game played over the internet by large numbers of players concurrently such as World of Warcraft[16] and EVE Online.[17] This type of game generally mixes a variety of gameplay types with multiplayer network-centric games in which large numbers of players cooperate and compete with each other on a large scale. The large scale poses significant technical challenges such as concurrency, fault tolerance, performance, persistence, and scalability. Employing an effective architecture can alleviate these challenges.

3.6. Software design

The process of *software design* deals with the instantiation (creation) of a DEG software design from the game architecture specification. The design process takes the game architecture specification document as input and produces a game software design specification document as the output work product.

DEG software is typically designed by using (a) object-oriented paradigm (OOP), (b) procedural paradigm (PP), or (c) combination of OOP and PP. Under the OOP, game software is designed by using objects instantiated from classes. Objects communicate with each other via message passing. Under the PP, game software is designed by using a main program calling subroutines at different logical points and subroutines calling other subroutines in a procedural manner.

The game software can be designed in one or more stages depending on the design complexity. Typically, we design complex game software in two stages, high-level design and detailed design or in three stages, preliminary design, interim design, and critical design.

Decomposition or modularization is the best solution for reducing and managing game software design complexity. A game software design is decomposed into modules (e.g. classes, subroutines) at level 1. Level 1 modules are further decomposed into modules at level 2. Level 2 modules are further decomposed into modules at level 3,

and so on. The decomposition is carried out horizontally or vertically and continues until the leaf modules are manageable in complexity.

We advocate the following design principle: Create a game software module in such a way that it has the highest possible cohesion and the lowest possible coupling. *Cohesion* refers to how much the elements included in a module are related to each other. *Coupling* measures the extent to which a module is constructed based on the logic of another module. Game software maintenance is facilitated and reusability is improved when the modules have high cohesion and low coupling.

Game software is designed by using software design patterns. A *software design pattern* is a reusable and community accepted solution to a commonly encountered software design problem. Many design patterns exist including creational patterns, structural patterns, and behavioral patterns.[18]

3.7. Programming

The *programming* process takes the game software design specification document as input and produces executable game software components as the output work product. The programming transforms a game software design into executable code by using a high-level programming language (e.g. C, C++, C#, Java, Objective-C, Swift) under an Integrated Development Environment (IDE), OpenGL (Open Graphics Library), or a game engine (e.g. Cocos2d, Unity, Unreal Engine). Components of a large complex game design can be programmed by different teams resulting in the production of multiple executable game software components.

Programming best practices such as the following should be employed: (a) meaningful class, method, and variable names, (b) effective source code documentation and in-line comments, (c) assertions used during testing and kept in the source code, but not compiled for the production release, and (d) effective source code formatting to improve readability.

3.8. Integration

Integration is the process of combining individually tested executable game software components developed by multiple teams into an integrated whole. This process takes the executable game software components as input and produces the integrated game software application as a finished product.

The process of integration should be well planned and executed. It is considered best practice to (a) establish an integrated product team consisting of one member from each component development team to oversee the planning and execution of the integration process, and (b) perform integration in a continuous and iterative manner, as opposed to all at once, to reduce the integration risk.

3.9. Publishing

Game software application can be published either as *Software as a Product* (SaaP) or *Software as a Service* (SaaS). Under SaaP, the game software is (a) made available either for download from a server computer (e.g. Apple App Store) or for delivery in a shrink-wrap box, (b) received by a user as a product, and (c) installed on a computer (e.g. desktop, laptop, handheld, or game console) for use. Under SaaS, the game software is (a) provided on a server computer, (b) connected to by a user over the internet, and (c) used remotely either free of charge or for a subscription fee (e.g. World of Warcraft,[16] EVE Online[17]).

3.10. Game-based learning

The *game-based learning* process refers to the process in which the completed game software application is used by students for learning a subject. The ultimate goal of a DEG is to provide a learning environment with the highest possible quality. During this process, that quality is assessed.

Assessment of game-based learning poses significant technical challenges for researchers and educators.[19] Available assessment techniques should be employed in this process for determining how well the students learn with a DEG and how effectively a DEG transforms learning in a way that game-based learning provides a better learning experience in comparison to other pedagogies. Ifenthaler et al.[19] indicate that ''Aligning learning and assessment is at the core of creating a favorable and effective learning environment, one that is learner-centered, knowledge-centered, and assessment-centered.''

3.11. Feedback

This process focuses on documenting the game-based learning assessment results and communicating them to the game developers as feedback. The game development restarts based on this feedback and goes through another life cycle iteration to produce a revised improved version of the DEG. This iteration is repeated to continuously improve the quality of the DEG during its life cycle.

3.12. Maintenance

The *maintenance* process, designated by a circle in Figure 2, starts after the DEG software application is published. It deals with the modification of a DEG software application after being published to (a) cope with changes in the DEG's external environment, (b) diagnose and correct faults, (c) improve quality by satisfying new or

changed user requirements, or (d) prevent problems in the future.

Four types of maintenance exist.

1. *Adaptive maintenance*: adaptations required as changes occur in the DEG's external environment (e.g. new operating system, new game engine, new screen sizes).
2. *Corrective maintenance*: fixing errors reported by the players and others.
3. *Perfective maintenance*: adding new capabilities, improving existing ones, or making functional enhancements based on feedback from the players and others.
4. *Preventive maintenance*: making modifications or reengineering so as to prevent potential future problems.

4. Concluding remarks

Success can be achieved in a complex game development project by employing a unifying methodology throughout its life cycle. This paper presents such a methodology named GAMED (diGital educAtional gaMe dEvelopment methoDology).

GAMED provides a road map (blueprint or detailed plan) for managing complex game development projects. It modularizes and structures game development into phases, processes, and work products within the framework of a life cycle. It presents principles, strategies, procedures, and step-by-step guidelines to follow. It advocates a quality assurance strategy throughout the entire game life cycle. Process by process, it provides guidelines to the developers and enables project managers to identify areas of expertise in which to employ qualified people. Key advantages of GAMED and its associated processes are that the risk of failure is reduced, quality is emphasized, inefficiencies are avoided, and the probability that the project is completed within budget and on time is increased.

GAMED is applicable for digital educational game development projects of all sizes. A small project may conduct some of the GAMED processes at a reduced level of effort and in a shorter period of time. GAMED is intended to provide guidance regardless of the game development project size. It is up to the user of GAMED to decide how much of its guidance to employ.

Although GAMED is created for digital educational game development, it can be tailored and effectively employed for the development of other types of games.

Funding

This work was supported in part by the National Science Foundation (NSF) Division of Research on Learning in Formal and Informal Settings (DRL) Discovery Research K-12 (DRK-12) program under Award Number 1118571.

References

1. Johnson L, Smith R, Willis H, Levine A and Haywood K. *The 2011 horizon report*. Austin, TX: The New Media Consortium, 2011.
2. Gee JP *What video games have to teach us about learning and literacy*. 2nd ed. New York: Palgrave Macmillan, 2007.
3. Prensky M. *Digital game-based learning*. St. Paul, MN: Paragon House, St. Paul, 2007.
4. Felicia P. (ed). *Handbook of research on improving learning and motivation through educational games: multidisciplinary approaches*. Hershey, PA: IGI Global, 2011.
5. Balci O. A life cycle for modeling and simulation. *Simulation: Trans Soc Model Simul Int* 2012; 88: 870–883.
6. Pressman RS and Maxim BR. *Software engineering: A practitioner's approach*. 8th ed. New York: McGraw-Hill Education, 2015.
7. Balci O and Nance RE. Formulated problem verification as an explicit requirement of model credibility. *Simulation* 1985; 45: 76–86.
8. SEI. *Capability maturity model integration (CMMI) for development, version 1.3*. Pittsburgh, PA: Software Engineering Institute (SEI), Carnegie Mellon University, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm
9. Zynga. Farmville. Zynga Inc., http://company.zynga.com/games/farmville
10. Boehm B. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engng Notes* 1986; 11: 14–24.
11. Sommerville I. *Software engineering,* 9th ed. Boston, MA: Addison-Wesley/Pearson Education, 2011.
12. Chigani A and Balci O. The process of architecting for software/system engineering. *Int. J. Syst. Systems Engng* 2012; 3: 1–23.
13. DoDAF. *DoD architecture framework ver. 2.0 vol. 1: Introduction, overview, and concepts: manager's guide*. Washington, DC: Architecture Framework Working Group, 2009.
14. DoDAF. *DoD architecture framework ver. 2.0 vol. 2: Architectural data and models: architect's guide*. Washington, DC: Architecture Framework Working Group, 2009.
15. DoDAF. *DoD architecture framework ver. 2.0 vol. 3: DoDAF meta-model physical exchange specification: developer's guide*. Washington, DC: Architecture Framework Working Group, 2009.
16. Blizzard. *World of warcraft*. Irvine, CA: Blizzard Entertainment, Inc., 2014, http://us.battle.net/wow
17. EVE. *EVE online*. Reykjavik, Iceland: CCP hf., 2014, http://www.eveonline.com
18. Gamma E, Helm R, Johnson R and Vlissides J. *Design patterns: elements of reusable object-oriented software*. Reading, MA: Addison-Wesley, 1995.
19. Ifenthaler D, Eseryel D and Ge X (eds). *Assessment in game-based learning: foundations, innovations, and perspectives*. New York: Springer, 2012.

Author biographies

Serdar Aslan is a PhD student in the Department of Computer Science at Virginia Tech. He received his BS degree in computer engineering from Kocaeli University (Turkey) in 2007, and his MS degree in computer science from Virginia Tech in 2011. His current areas of expertise center on mobile software engineering and game development.

Osman Balci is a professor of computer science and director of the Mobile/Cloud Software Engineering Lab at Virginia Polytechnic Institute and State University (Virginia Tech). He received his BS and MS degrees from Boğaziçi University (Istanbul) in 1975 and 1977, and MS and PhD degrees from Syracuse University (New York) in 1978 and 1981. Dr Balci currently serves as editor-in-chief of *ACM SIGSIM Modeling and Simulation (M&S) Knowledge Repository*, M&S category editor of *ACM Computing Reviews*, and editorial board member of several journals. He served as the founding editor-in-chief of two international journals: *Annals of Software Engineering*, 1993–2002 and *World Wide Web*, 1996–2000. He served as chair of ACM SIGSIM, 2008–2010; and director-at-large of the Society for Modeling and Simulation International (SCS), 2002–2006. Dr Balci received the 2013 McLeod Founder's Award for Distinguished Service to the Profession given by SCS. His current areas of expertise center on iOS mobile software engineering, cloud software engineering, modeling and simulation, and digital game-based learning. His e-mail and web addresses are, respectively, balci@vt.edu and http://manta.cs.vt.edu/balci.