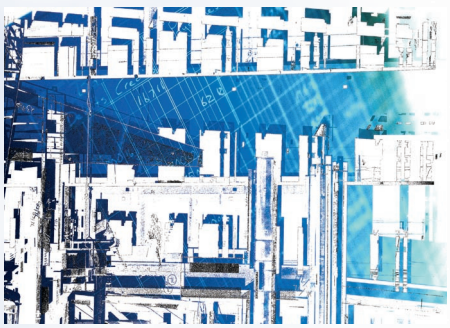
A Content-Centric Development Process Model

Working from the belief that when content is king, content experts should lead, a storyboard-driven approach provides a sound methodology for developing educational games that helps ensure that no good storyboard becomes a bad game.



Pablo Moreno-Ger, Iván Martínez-Ortiz, José Luis Sierra, and Baltasar Fernández-Manjón
Complutense University of Madrid

Computer and videogames have a huge potential to facilitate learning because they easily capture students' attention. The popular notion that children have a limited attention span falls apart when we see that they can spend hours playing a game without losing their concentration. The time spent gaming also can be an educational investment if applied carefully—and that is where game-based learning becomes relevant.

However, it's not easy to educate while entertaining. We can't just throw some math into a game and call it educational, nor can we call it entertainment when a talking animal gives the lesson while the student solves silly puzzles. In the words of literature professor Henry Jenkins, we need to "move beyond the current state of edutainment products which combine the entertainment value of a bad lecture with the educational value of a bad game" (http://icampus.mit.edu/projects/GamesToTeach.shtml).

In our view, point-and-click adventure games like the *Monkey Island* or *King's Quest* sagas have all the ingredients needed to achieve this balance between content delivery and entertainment. In this genre, we measure games in terms of their storyboards' quality and their rhythm as opposed to the typical features of other videogame genres, such as challenges that make players' adrenaline flow or demand lightning reflexes.

A narrative game in which the content is pervasively woven into the storyboard has the potential to achieve this elusive balance. However, the participation of nontechnical professional scriptwriters in the videogame industry has always raised issues in terms of their integration in technical development teams. In our case, the outlook is even worse because the writing teams include experts in the subject matter that the game aims to teach, and they would likely feel uncomfortable surrounded by developers. As developers and computer science instructors, we see how technological constraints often guide our design choices regarding the user experience.[1] When applied to adventure game development, this can cause conflicts and

Published by the IEEE Computer Society
0018-9162/08/$25.00 © 2008 IEEE

prompt the technical team to make statements such as "look, this part of the storyboard is great, but it can't be done with our game engine/programming language" usually due to some obscure reasons the writers don't understand. These situations generate friction and can hinder the development process if the participants understand it as a confrontation between programmers and writers.

Thus, in our educational scenario, there is a direct need for a well-defined development process model that seamlessly includes game writers and instructors in a traditional game development organization while maintaining their work independent of any technological requirements. As the "Approaches to Game Development" sidebar describes, developers can take any of several approaches to designing a game, but none of them places the writers in the center of the process. We advocate a development process model in which the game writers know precisely what can and can't be done with the current language because they are the language's final users. With this approach, even if the development language is essential, the storyboard still drives the development. This process embraces change by having the language evolve along with the game. For this purpose we conceived the <e-Adventure> development model: The game's writers lead the process, and the documents they create provide the keystone for the entire project.

DOCUMENTAL APPROACH

The cultural clash between instructors, game writers, and developers is common to the development of other content-intensive applications such as hypermedia and educational programs. Developing a content-intensive application requires collaboration between experts in the content domain and the programmers who will build the basic functionality to display and process that content. Since field experts usually have no technical background, we must facilitate their task as much as possible. Otherwise, their attention shifts away from what really matters: their knowledge of the field.

Our documental approach to software development proposes a collaboration model in which domain experts and developers work together using documents that describe the application's contents and other relevant features. Developers equip domain experts with a suitable markup language and an application generator. Domain experts mark up the documents with the language and process them automatically with the generator, yielding the final executable application.[2]

In a basic sense, the application of this approach to educational game development can be understood as an educational game engine that uses an XML notation to describe the games. In this simplified conception, the game writers act as clients of the engine, adjusting ideas to the tool's characteristics. Even though this approach is useful for amateur developments or fast prototyping, the functionality of these environments becomes a factor that limits creativity because the language constrains what can and can't be done.

As long as a direct relationship exists between the environment's complexity and its expressive power, easier languages can impose severe limitations. Thus, defining the documental approach to software development as a language-engine game-development pattern grossly simplifies the concept.

The documental approach manages the roles and responsibilities of domain experts and developers without turning the former into clients and the latter into providers. On the contrary, it envisions the development process as a collaborative effort led by the writers, where the language is constantly evolving to fit the storyboard's needs and the final versions of the application, language, and engine are obtained simultaneously.

Approaches to Game Development

Videogame development poses a complex task that requires highly specialized skills in areas such as graphics processing, animation, sound, and buffering. Game developers must take different approaches to dealing with the additional complexities that are present compared to traditional programming:

- Use of a general-purpose programming language such as C++ in combination with tools that facilitate low-level graphics processing such as DirectX or OpenGL.
- Use of a domain-specific game-oriented programming language such as Dark Basic or an authoring tool that lets nonprogrammers develop their own games in specific genres, such as Game Maker.[1]
- Use of a game engine that manages all low-level tasks, such as graphics, animation, collision detection, sound, and AI. The game's writers and designers configure the engine using a scripting language that is simpler than a general-purpose programming language. Developers use this approach most frequently because the set of skills required to design interesting gameplay is not the same as that required to implement a highly optimized graphical pipeline.

Reference

1. M. Overmars, "Teaching Computer Science through Game Design," *Computer*, Apr. 2004, pp.81-83.

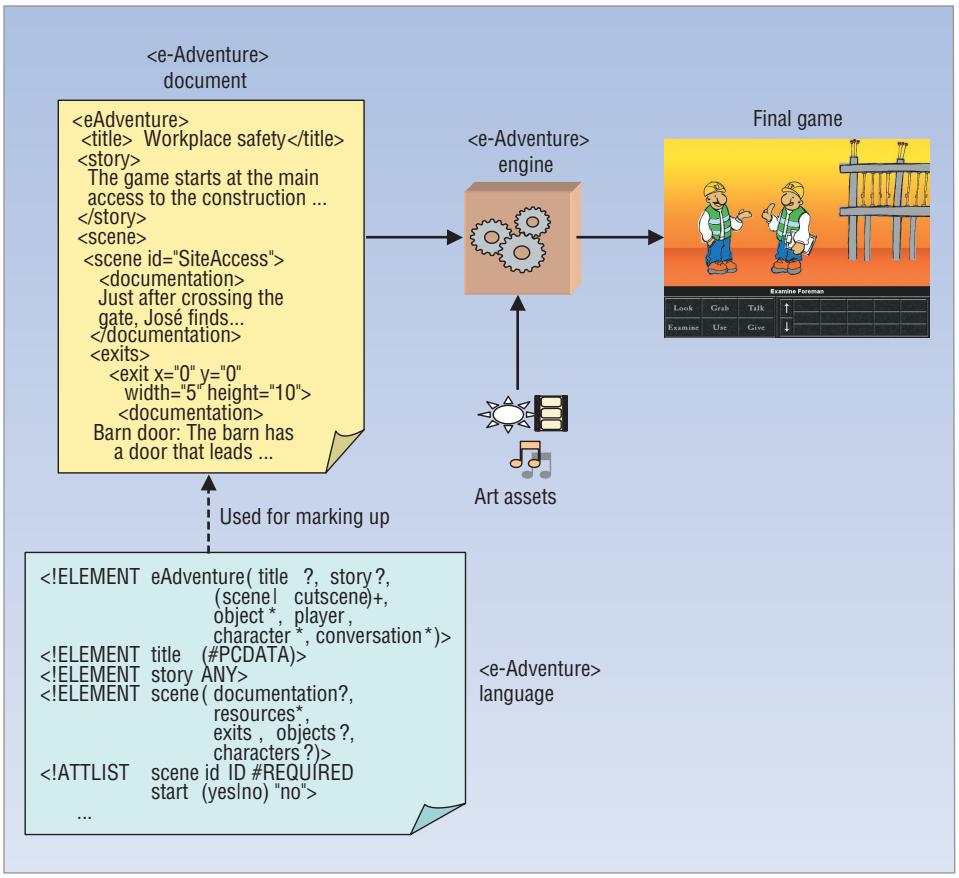


Figure 1. <e-Adventure> products. Each development iteration delivers products closer to the desired final result, including the language, document, art assets, engine, and final videogame.

Products

Products in the <eAdventure> process include the language, document with the marked-up storyboard, art assets, engine, and final game. These products play the following roles:

- Following our documental approach, we envision the <e-Adventure> language as a descriptive domain-specific markup language defined as an XML application. This language will closely mirror the structure of typical storyboards for adventure videogames. Thus, it will include element types for scenes (such as a living room, pub, or square), cut scenes (fixed places that include special events in the game flow, such as playing a video), characters, objects, and conversations.
- The <e-Adventure> document provides an XML description of the game's storyboard. This document conforms to the <e-Adventure> language.
- The art assets supply all the multimedia materials required to render the final game, such as background images and character and object sprites.
- The <e-Adventure> engine interprets the <e-Adventure> language. When fed the game's storyboard and art assets, it generates the final graphical adventure videogame.

<E-ADVENTURE> PROCESS MODEL

We developed <e-Adventure> in collaboration with the Spanish National Center of Information and Educative Communication (CNICE). This office of the Spanish Ministry of Science and Education contains Spain's largest repository of educational computer-assisted material written in Spanish, including interactive videogames to support K-12 education. <e-Adventure> (formerly known as <e-Game>) applies the documental approach to developing educational adventure games.[3] In its basic conception as a game development language, it would be the educational equivalent of products such as Adventure Game Studio (www.adventuregamestudio.co.uk), although still in a prototype stage. It includes several pedagogical enhancements such as built-in assessment, standardized educational metadata, and integration with virtual learning environments.

However, <e-Adventure>'s most interesting feature is its underlying process model. We base this model on spiral or iterative process models such as the Rational Unified Process (RUP),[4] dividing the process into several stages that can be iterated as often as necessary. Each iteration involves both technical and nontechnical stakeholders, who deliver products closer to the desired final result.

In <e-Adventure>, the supporting tools—the language and associated processor—are organic entities that constantly evolve along with the game in a process of producing and maintaining the XML documents with the storyboards and corresponding art assets. Thus, each iteration provides a new version of the game, a refined version of the language, and its corresponding implementation. While modifying the supporting tools at each step might be considered bad practice from a software engineering perspective, this is not true in every case. Borrowing from a growing trend in corporate environments, we consider the supporting tools as *probably flawed* and embrace change by explicitly introducing the concept of change management into the process.

When the language can't express a particular concept, we try to learn from this instead of developing a workaround. This kind of planning results in the additional task's cost being relatively low. On the other hand, the extended practice of considering the supporting tools as immutable can be quite costly should there be a need to change them in the middle of a process that doesn't anticipate or readily allow for change.

Thus, even if we have artifacts such as the <e-Adventure> language and engine, when our development process goes active, both the language and engine grow and evolve during the process, responding to new script requirements because one process stage explicitly addresses this objective. Developers evaluate the modifications and additions at the end of the process to decide which should remain for future development and which are specific and thus should be discarded.

Participants and their roles

Even though instructors might use a specific version of the language and engine to develop small educational games, we can target the process model toward larger development teams with mixed roles and skills. In particular, we foresee four main stakeholders in the <e-Adventure> process: scriptwriters, programmers, artists, and project supervisors.

Scriptwriters draft the games' scripts and, depending on the project's goals, this group might be formed by professional writers, instructors, or both. In any case, we think this group is key to the final product's success in terms of both entertainment and educational value—and for that reason it receives most of the attention during the production process. Since these stakeholders are the <e-Adventure> language's final users, their input is crucial to its evolution and ability to frame their requirements in functionality and usability terms.

Programmers oversee the creation, fine-tuning, and specialization of the game engine to accommodate the particular needs of each production by, for example, providing a specific component to render a complex scene, to play a particular media type in a cut scene, or to add a new characteristic to the engine. In addition, their background generally includes skills related to language processing, thus they too have a stake in the language's evolution.

Artists produce the art assets. While in simple productions the instructors or game writers can take the art assets from repositories, or even create them on their own, in <e-Adventure> we explicitly include artists—graphic designers, musicians, and so on—as distinguished stakeholders in the development process.

Project supervisors manage the project's evolution. In particular, they have the final word on approving or rejecting both the initial game storyboard and resulting game. They must also keep track of the process and assess the results of each iteration. Depending on the project's scale, this group might or might not be explicitly present.

Production process

Figure 2 shows the production process itself, which begins with the conception of the storyboard's first draft (Conception of the Storyboard activity). This activity is the storywriters' main responsibility. Once supervisors approve the storyboard's draft (Revision checkpoint), they hold a meeting between writers and programmers to decide whether the <e-Adventure> language has enough expressive power to implement the storyboard (Evaluation checkpoint) in its current state. If they detect flaws in the language, they can customize it by extending and adapting the existing markup structures or adding new structures to the language (Language Customization activity). This step is crucial in our change-management routine, as it puts the spotlight on the script and not on the underlying technology.

Equipped with the initial storyboard and the <e-Adventure> language, scriptwriters, programmers, and artists undertake the actual production of the artifacts.

Equipped with the initial storyboard and the <e-Adventure> language, scriptwriters, programmers, and artists undertake the main step in the process—the actual production of the artifacts that come out of this iteration. Scriptwriters use the new version of the language to encode the storyboard or refine the existing marked storyboard, possibly with the help of the programmers to clarify the language's most complex constructs. Additionally, programmers modify the engine to improve or fine-tune it or to process the new syntactical constructs. The artists use the storyboard to identify which assets are required for its implementation, then provide those assets.

Finally, developers combine all these artifacts to yield an executable version of the game (Game Production). Producing the running game provides the milestone that marks the iteration's end. The project supervisors evaluate the products and establish the guidelines for the next iterations, including changes in the script, rejection of changes, reevaluation of the project's schedule, and so on. In the final iterations, this evaluation might include tests with users or the intervention of a quality assurance department.

After this evaluation, a new iteration begins. However, before producing more assets or writing more lines of code for the engine, the developers reevaluate the language and storyboard. During the implementation phase it is normal to find that parts of the storyboard lack polish, to discover new ideas that can't be implemented

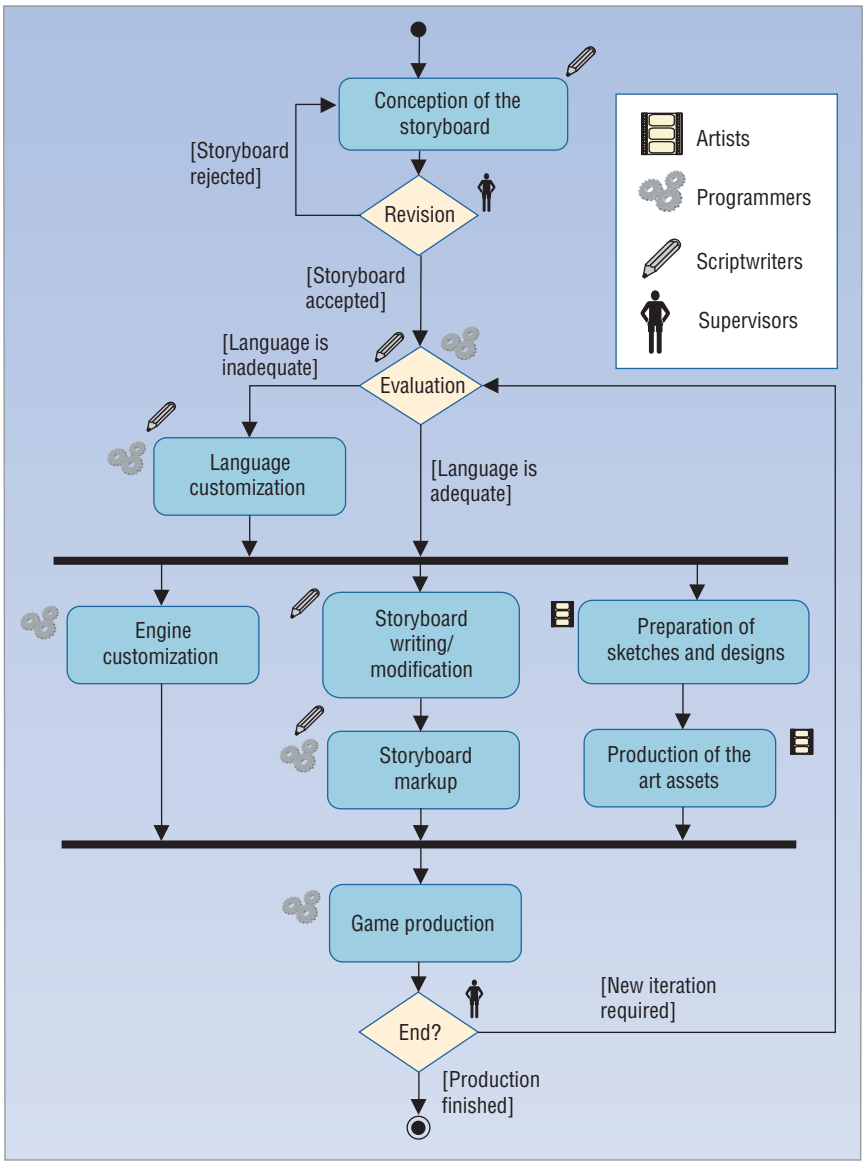


Figure 2. <e-Adventure> production. The process begins with the scriptwriters' main responsibility, the conception of the storyboard's first draft. Once supervisors approve this draft, the writers and programmers meet to decide whether the <e-Adventure> language has enough expressive power to implement the storyboard in its current state.

with the current version of the language, and even to identify things that seemed like a great idea before but, when implemented, don't seem to work. After this step, there will be a new specification for the language and the engine, a new version of the storyboard, and the need for more art assets. At this point, the new implementation phase can start.

As with the RUP, completing all stages of each iteration does not imply that each iteration ends with a potentially deliverable product. The first iterations essentially focus on creating a solid storyboard and establishing the language's major features. The implementation stage of these early iterations creates prototypes of the engine, explores whether the language can be used in practice to implement the storyboard, and creates several predesigns of the art assets to start exploring what the general aesthetic of the game will be.

PROCESS INTO PRACTICE: *SCHOOLWORKS*

The *Schoolworks* educational game shown in Figure 3 exemplifies the kind of project that can benefit from the <e-Adventure> approach. Written as an initiation module for a course on safety regulations in construction, in this game the player assumes the role of a novice construction worker recently hired to help build a new school, as Figure 3a shows. Over five work days, the player receives several assignments that must be performed at the construction site, each of which requires following several safety regulations. The tasks include dealing with hazards related to falls, misuse of toxic materials, electrical shocks, and physical injuries while handling heavy objects.

The game's point-and-click adventure structure doesn't lead the player step by step into memorizing all the regulations. Instead, the player is assigned different tasks that require following those regulations. Every time the player violates a regulation, a mishap occurs, and the player's avatar is either injured or fired.

As previously mentioned, a rich story that attracts the player is key to the game's success. Ideally, the project succeeds by weaving this content into the story. Thus, along with the construction tasks, in *Schoolworks* the player experiences a compelling narrative, with several colorful characters such as the patronizing foreman or the crazy retired construction worker who roams the site. During the game, a secondary story unfolds when the player discovers that the construction company plans to cancel building at the school and have the land rezoned for a mall. Discovering this, the player eventually joins the group opposing this plan.

This project employed a sizable development team compared to other educational developments, although still smaller than the teams for top-notch commercial videogames. Three people formed the writing team, two of whom possessed a background in writing noneduca-

tional adventure games, while the third had been an instructor on the subject.

Even if they were not trained programmers, all three writers could be considered computer literate. The programmers were computer science students who possessed broad experience with both XML technologies and game engine programming, while the artists were formal art students enrolled in a computer animation course.

The team approached the game as an evolution of a previous proof-of-concept development. Their first step consisted of reusing the ideas included in the first game and adding the necessary narrative spice to make the story more appealing. This resulted in a storyboard that required approval by the supervisors who were the corresponding leaders of the three teams involved. After initial approval, the iterative development process began.

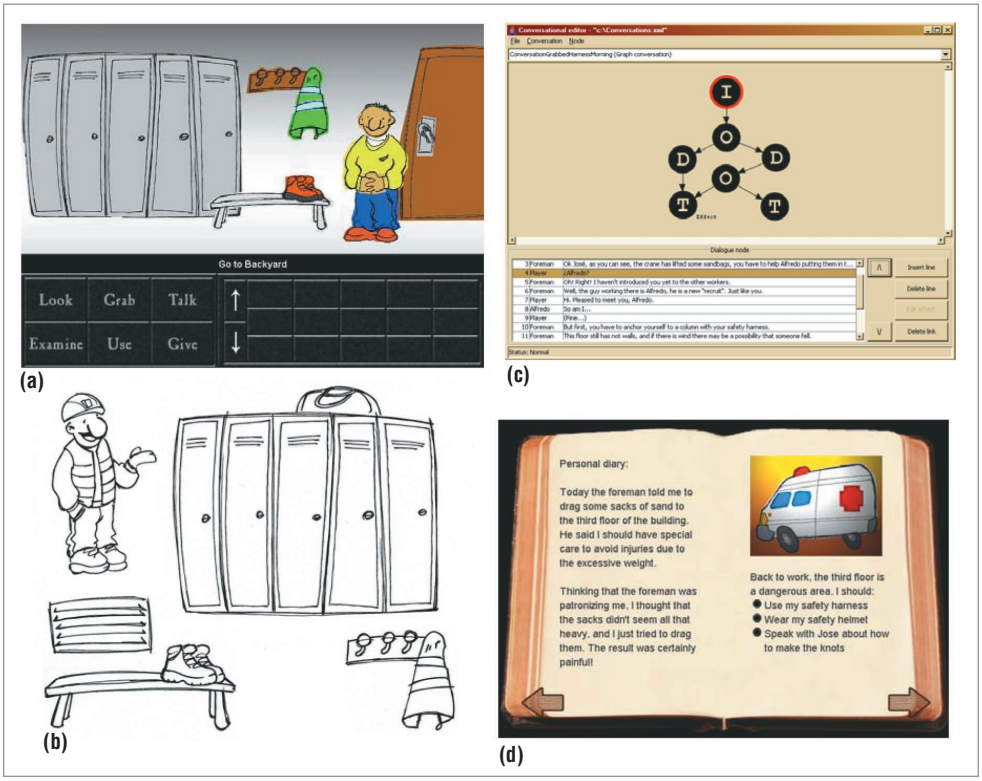


Figure 3.* Schoolworks *project. Three people formed the writing team, two of whom possessed a background in writing noneducational adventure games, while the third had been an instructor on the subject. (a) Educational game; (b) early sketches; (c) graphical tool for creating graph-shaped conversations; (d) in-game book.

First iteration

First, the team determined if the language had the power to express the storyboard's contents. After reviewing the requirements, we noticed that some of the planned in-game conversations would pose problems when fitting into the treelike structure of the conversations that the language supported. The programmers suggested a new mechanism to define graphlike conversations, which the team accepted.

At the implementation stage, scriptwriters added descriptive markup to the most critical portions of the game, producing the storyboard's first XML version, which the rest of the work would focus on. In turn, the programmers modified the engine by adding support for the new conversation format, while at the same time helping the scriptwriters with the most complex parts of the markup process. Meanwhile, the artists used the original draft to prepare some paper-based sketches that defined the game's aesthetics—as shown in Figure 3b—by proposing a cartoonlike style that the team accepted.

Second iteration

After the first iteration, the supervising committee met again to review the process and start assessing the language's limitations. Scriptwriters reported that the syntax for graphlike conversations had proven far too complex and hindered the markup process. Since nobody wanted to go back to treelike structures, the programmers suggested developing a graphical tool to create these conversations and then generate the corresponding XML automatically, as Figure 3c shows.

During the corresponding implementation phase, programmers implemented the graphical tool to edit conversations, scriptwriters marked up most of the game, and artists created the most important art assets and placeholder graphics so that a preliminary version could be executed and tested. This iteration ended with the first game play tests and evaluations.

Third iteration

During testing, we quickly identified that players would spend most of their time dividing their attention between the screen and the books containing the regulations. To improve this, the scriptwriters proposed using in-game books, as Figure 3d shows. These books contained summaries of the regulations that the player could bring up at any moment during game play. The develop-

ers enhanced the language with syntax to define in-game books, supporting text, headers, and images.

In the final implementation phase, the scriptwriters added the books and completed the entire descriptive markup. Programmers implemented support for the books and introduced several performance enhancements in the engine's implementation without affecting the language. In turn, the artists provided the final art assets, including animations, backgrounds, cut-scene videos, and background music.

Maintenance and tweaking

After completing the previous stages, the team upgraded the final products: Enhanced versions of the <e-Adventure> language, <e-Adventure> engine, art assets, and marked-up storyboard. As usual in the development process, they automatically realized the game by feeding the engine the storyboard and art assets. At this point, the other main advantage of the documental approach became evident: Going forward, artists and programmers are no longer needed. So long as writers don't require new linguistic constructs, they can modify the game, correct errors, improve conversations, translate the content into another language, or even adapt it to fit different regulations in different countries—all without requiring the programmers' assistance. The marked documents maintain their original storyboard structure and are human-readable. Thus, anyone with basic computer skills can tweak the game and improve it.

The storyboard-driven approach provides a sound methodology for developing games that have as their keystone the final product's content. Adventure games focus on content. When we add the educational goal into the mix, this becomes even more relevant. From this notion, instead of adapting the content to fit the technology, we adapt the technology to fit the content. In other words, when content is king, content experts should lead.

Even though the resulting process might seem burdensome compared to more streamlined processes, this approach has a specific focus that ensures that no good storyboard becomes a bad game.

Acknowledgments

The Spanish Committees of Education and Science/Industry supported this work (TIN2007-68125-C02-01, TIN2005-08788-C04-01, FIT-360000-2007-23). The Complutense University of Madrid, the regional government of Madrid, and Santander Bank also supported this work (FPI 4155/2005, UCM 921340, and Santander/UCM 2007/1725). Thanks to the Spanish National Center of Information and Educative Communication for providing game design documents and art assets.

References

1. A.R. Cooper, *The Inmates Are Running the Asylum: Why High-Tech Products Drive Us Crazy and How to Restore the Sanity*, Macmillan Computer Publishing, 1999.
2. J.L. Sierra, A. Fernández-Valmayor, and B. Fernández-Manjón, "A Document-Oriented Paradigm for the Construction of Content-Intensive Applications," *The Computer J.*, vol. 49, no. 5, 2006, pp. 562-584.
3. P. Moreno-Ger et al., "A Documental Approach to Adventure Game Development," *Science of Computer Programming*, vol. 67, no. 1, 2007, pp. 3-31.
4. P. Krutchen, *The Rational Unified Process: An Introduction*, 3rd ed., Addison-Wesley, 2003.

Pablo Moreno-Ger *is a member of e-UCM (www.e-ucm.es), the e-learning research group at the Complutense University of Madrid (UCM). His research interests include e-learning technologies and the integration of videogames and simulations in learning environments. Moreno-Ger received a PhD in software engineering and artificial intelligence from UCM and is leading development of the <e-Adventure> initiative. Contact him at pablom@fdi.ucm.es.*

Iván Martínez-Ortiz *is a lecturer in the Department of Software Engineering and Artificial Intelligence at UCM. He is a member of the e-UCM group. His research interests include e-learning technologies and the integration of educational modeling languages and workflow technologies. Martínez-Ortiz is a PhD candidate in software engineering and artificial intelligence. Contact him at imartinez@fdi.ucm.es.*

José Luis Sierra *is an associate professor in the Department of Software Engineering and Artificial Intelligence at UCM and a member of e-UCM. His research interests include e-learning technologies, domain-specific languages, and markup languages. Sierra received a PhD in computer science from UCM. Contact him at jlsierra@fdi.ucm.es.*

Baltasar Fernández-Manjón *is an associate professor in the Department of Software Engineering and Artificial Intelligence at UCM. He is also the vice dean of Research and Foreign Relationships at the university's Computer Science School and codirector of the e-UCM group. His main research interests are e-learning technologies, educational uses of markup technologies, application of educational standards, and user modeling. Fernández-Manjón received a PhD in physics from UCM. Contact him at balta@fdi.ucm.es.*