Requirements Engineering and the Creative Process in the Video Game Industry

David Callele, Eric Neufeld, Kevin Schneider
Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada S7N 5C9
{callele,neufeld,kas}@cs.usask.ca

Abstract

The software engineering process in video game development is not clearly understood, hindering the development of reliable practices and processes for this field. An investigation of factors leading to success or failure in video game development suggests that many failures can be traced to problems with the transition from preproduction to production. Three examples, drawn from real video games, illustrate specific problems: 1) how to transform documentation from its preproduction form to a form that can be used as a basis for production, 2) how to identify implied information in preproduction documents, and 3) how to apply domain knowledge without hindering the creative process. We identify 3 levels of implication and show that there is a strong correlation between experience and the ability to identify issues at each level.

The accumulated evidence clearly identifies the need to extend traditional requirements engineering techniques to support the creative process in video game development.

Keywords: Non-functional requirements, elicitation, video game development, game design document, preproduction, production, domain-specific terminology.

1 Introduction

Video games are a special type of multimedia application – an entertainment product that requires active participation by the user. Developed by a multi-disciplinary team, non-functional requirements such as entertaining the user create special demands on the requirements engineering process. Requirements like *fun* and *absorbing* are not well understood from the perspective of requirements engineering, compounding communication issues between game designers and software engineers. Game designers may not understand, for example, the limitations of artificial intelligence when designing non-player characters while software engineers may not understand the creative vision or they may be too willing to compromise that vision in the rush to ship the product.

It may be that nothing can qualitatively change this. However, it should be possible to decrease the cost of delays caused by communication errors in such a heterogeneous group. As a first step toward the development of a formal process, we have attempted to locate the causes of the most costly errors. By way of background, we first review the requirements engineering literature applied to multimedia development and introduce the video game industry and the video game development process, with attention to the roles of preproduction and the game design document (as a deliverable artifact of the preproduction process). We analyze the observational reports from the Postmortem column in Game Developer magazine, categorize the information therein, and present the results. Three examples, drawn from real video games, illustrate particular issues that must be addressed in a formal process. We follow with our conclusions, an analysis of the role of requirements engineering in video game development, and directions for future work.

2 Background

Requirements engineering within a community of common interest is difficult – the ability to precisely communicate and capture stakeholder wants and needs is rare. Traditional requirements engineering techniques [15, 26] assume these communications issues can be overcome in a few iterations. However, we are unaware of any work that directly addresses the validity of this assumption in a multi-disciplinary development effort. While goal [1, 10] and scenario [16] based techniques can be used to alleviate communications issues, their efficacy when development efforts include a strong artistic or inventive element [25] (such as in video game design, multimedia web sites, or the movie industry) remains unproven.

Members of video game development teams include practitioners from such diverse backgrounds as art, music, graphics, human factors, psychology, computer science, and engineering. Individuals who, in other circumstances, would be unlikely to interact with each other on a professional basis unite in their economic goal of creating a commercially successful product. Requirements engineering in the face of such diversity requires the creation of a common (domain) language (and implied world model) specific to the task at hand. Once all stakeholders fully commit to the domain language, then a set of requirements that captures the stakeholders wants and needs can be generated.

Given the dearth of directly related work, we performed a more extensive literature review, focussing on: (1) requirements engineering and emotional factors (including fun in games), (2) issues of language and the creation of a common language or domain ontology, and (3) requirements elicitation and the effects of feedback on emergent requirements, particularly in multimedia development.

2.1 Emotional Factors

While emotion in human-computer interaction is coming under ever increasing scrutiny [20], few researchers have investigated emotional factors in requirements engineering. Draper [11] looked at fun as a candidate software requirement, attempting to identify what it is that makes play *fun*. He concluded that "fun is not a property of software, but a relationship between the software and the users goals at that moment" and that "providing enjoyment is now a defining requirement of an important class of software, and this has not been sufficiently recognized in our analyses and design methods". These conclusions are consistent with our experience.

Hassenzahl *et al.* [17] introduced *hedonic* qualities (those that are unrelated to the current task but present for emotional reasons) and associated repertory grid techniques for measuring them. Bentley *et al.* [4] investigated emotional (affective) factors in computer games, noting that "software requirements for these and other affective factors are never truly captured in an official manner". In particular, usability, immersion, and motivation were considered via a user survey mechanism. They note that there are no established techniques for eliciting emotional requirements. Even Chung, in his detailed analysis of non-functional requirements [8], does not substantively address emotional issues.

At their best, video games stimulate a state of *flow* in the player, engendering concentration so intense that their perception of time and sense of self become distorted or forgotten [9]. In the field of game design, Salen and Zimmerman [28], Laramee [19], and Saltzzman [29] address issues of emotions and emotional response in game players. While these works do not directly address requirements engineering practices, the techniques that they describe for game design and eliciting feedback from players may increase the range of elicitation techniques available to practitioners. In a more general sense, Norman [23] describes numerous human factors practices that could be readily incorporated into requirements engineering for video games.

2.2 Language and Ontology

Zave [31] classifies the problems addressed by requirements engineering, defining the domain, in part, as "...translation from informal observations of the real world to mathematical specification languages." In game development, this is only partially true. In many cases, the game designer, an individual who may have little or no interest in a mathematical representation, is also tasked with generating the requirements. Unable to generate the requirements in isolation, the game designer works with the production team to translate the vision to requirements – usually stated in natural language complete with domain specific terminology. Once captured, the requirements may be formalized in place or, more likely, formalized as they are translated into specifications.

A common language [14, 30], ontology [18, 30, 7], or vision [18] is often mentioned as the solution to communications issues between disparate stakeholders. Natt och Dag *et al.* [24] have demonstrated the application of statistical natural language processing techniques to managing and understanding requirements generated by a multitude of sources. Their results may be applicable to the documentation transition issues studied further in Section 6.1.

2.3 Elicitation, Feedback and Emergence

Goguen [14] emphasizes that "feedback and feedforward go on all the time, at least in successful large projects" and that "requirements are emergent". Emergent requirements discovered during the transition from preproduction to production are a significant aspect of the creative design process.

Zave [31] presents a classification scheme that assumes that "...as software engineers, we can seek to understand social factors but we can only hope to influence technical practices." We posit that requirements engineering can be more proactive in video game development by providing feedback from production to preproduction in response to a feedforward of early versions of preproduction documentation. The resultant influence on the creative process escapes Zaves technical practices restriction. Specific feedforward and feedback examples appear in Section 6.3.

3 Video Game Development

Video games are a significant element of the entertainment industry. The Consumer Electronics Association [2] reports that entertainment software sales rose from $5.1 billion in 1999 to $7.7 billion in 2003 and that hardware sales increased from $2.3 billion in 1999 to $3.2 billion in 2003. Combined hardware and software sales in the video game industry exceed the 2003 $9.42 billion gate receipts of theatrical release movies in North America [13].

However, for every advertisement for a newly released game, the trade press reports a disproportionately large number of projects that fail to reach the market. The present work begins an investigation into the causes of these failures. The multidisciplinary nature of the video game development process – with art, sound, gameplay, control systems, human factors (and many others) interacting with traditional software development creates complexities that may recommend a specialized software engineering methodology for this domain.

3.1 Development Process

Figure 1 models the game development process as two consecutive efforts. The left hand side of the diagram depicts the preproduction phase, resulting in a Game Design Document (GDD). Preproduction loosely corresponds to a customers internal efforts to define their wants and needs before meeting with the development team.

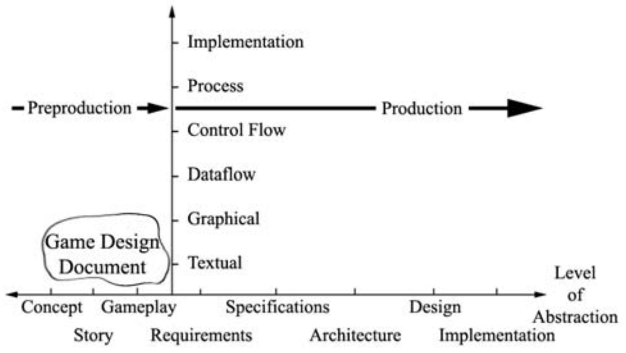


Figure 1. Video game development

The right hand side of the diagram, derived from Medvidovich and Rosenblum [21], depicts the production phase. Requirements engineering, with the assistance of the game designer(s), transforms the GDD to a specification (see Section 3.2). Once the specification is complete, a traditional software development process begins (often using an iterative development effort of some form), resulting in the game artifact.

Moving from preproduction to production is particularly difficult in video game development. A wide range of factors (*e.g.* artistic, emotive, and immersive factors) must be addressed by the requirements engineering effort. These factors are captured in the game design document.

3.2 The Game Design Document

The game design document is a creative work written by the game designer (or game design team). The GDD must be thorough, but not necessarily formal (in the sense of structure or from a mathematical perspective). In fact, one could argue that imposing too much structure on the creative process may be highly detrimental – constraining expression, reducing creativity, and impairing the intangibles that create an enjoyable experience for the customer. In a sense, the GDD is the requirements document as defined by the preproduction team.

The form of the game design document varies widely across genres and studios. Typically, a GDD (drawing loosely from Bethke [5]) includes a concept statement and tagline, the genre of the game, the story behind the game, the characters within the game, and the character dialogue. It will also include descriptions of how the game is played, the look, feel, and sound of the game, the levels or missions, the cutscenes (short animated movie clips), puzzles, animations, special effects, and other elements as required.

A game design document is a preproduction artifact designed to capture a creative vision. It is not designed to meet the needs of a production effort. If a GDD is being used as a source document in the production phase, there are two possible explanations. The game design document may contain the information required for the production phase. In this case, the game design document is malformed and should be restructured and maintained as independent preproduction and production documents. Or, it may be that, even though the game design document does not contain production information, the production team is performing requirements engineering, specification, and possibly even design, on an *ad hoc* basis. The greatest danger associated with such *ad hoc* activities is the dependence on human memory for capturing decisions and their justifications.

There are issues associated with managing the game design document to requirements document transition. Two sets of documentation must be created and maintained. The writing styles associated with the two sets of documentation are very different – is it reasonable to expect that a single individual can perform both tasks in an efficient and acceptable manner, particularly in the absence of generally accepted practices for performing this translation? In general, we found little evidence of structured application of generally accepted requirements engineering principles in our review of observational reports on industrial practices

(Section 5).

4 The Transition from Preproduction to Production

Requirements errors are some of the most costly to fix; Boehm and Basili [6] estimate that errors of this type can cost up to 100 times more to fix after delivery than if caught at the start of the project. Despite the available evidence and accumulated experience, many projects still suffer from failures due to inadequate requirements engineering.

Game designer and producer Eric Bethke [5] states

> . . . too many projects violate their preproduction phases and move straight to production. . . . In my opinion, preproduction is the most important stage of the project. I would like to see the day when a project spends a full 25 to 40% of its overall prerelease time in preproduction. During production there should to be relatively few surprises.

He promotes the use of UML based tools as a way to manage the transition but a formal (or semi-formal) transition process is not presented. Many of the requisite elements for production management (such as requirements capture, requirement analysis, task analysis, time estimation, project plans and technical design) are discussed in an informal manner.

Other producers and consultants, such as Rollings [27] and Michael [22], also identify many of the requisite elements for production management but do not provide formal or semi-formal guidelines for managing the transition.

When discussing game design documents, Bethke [5] comments ". . . I have never seen a completed design document, and one of the reasons is that game design documents need to be maintained through the course of production." With time-to-market pressures so prevalent, it is easy to see how documentation maintenance is given low priority.

Despite the recognized need, we have discovered no evidence that a process for managing the transition from preproduction to production has been proposed (recognizing that such a process may exist within an organization but remain unreported in the literature).

5 Review of Postmortem Columns

The video game industry is competitive and management processes are significant corporate assets and generally inaccessible to the researcher. Therefore, we use the *Postmortems* columns in Game Developer magazine [3], some of which are extracted in *POSTMORTEMS from Game Developer* [12], as a source of observational reports on this issue.

From the authors guide provided by the publisher:

> . . . Explain what 5 goals, features or aspects of the project went off without a hitch or better than planned. . . . Explain what 5 goals, features or aspects of the project were problematic or failed completely. . . . Important: try to come up with things that went right/wrong during project that are likely unique to your project. Stay away from common and well understood problems and solutions (*e.g.*, "communication between the team members wasnt good – thats been true of most games), and focus on what made your project different from others.

The reports presented in the Postmortem column potentially capture what makes video game development unique. They are typically attributed to members of the project management team or middle to upper management within the development organization. As such, one can reasonably assume that the reports reflect issues of particular import to the authors. While there may be an observer effect, particularly with respect to those items that went wrong, we assume the information presented has a strong basis in fact.

Fifty postmortem reports [3], published between May 1999 and June 2004, were analyzed in an attempt to identify factors that lead to success or failure in video game development. Each report contained 5 entries in the "what went right" and "what went wrong" sections. These entries were reviewed and classified according to the following scheme[1].

The classifications scheme has five categories: (1) *preproduction*, issues outside of the traditional software development process such as inadequate game design or inadequate storyboarding, (2) *internal*, issues related to project management and personnel, (3) *external*, issues outside of the control of the development team such as changes in the marketplace and financial conditions, (4) *technology*, issues related to the creation or adoption of new technologies, and (5) *schedule*, issues related to time estimates and overruns. Schedule issues are a subset of internal issues, but were uniquely identified in an effort to determine if scheduling was a significant issue. Any pair of the five categories was also possible (*e.g.*, "internal and technology") if the entry was that precise.

Figure 2 is a normalized representation of the results of the categorization process; of the 15 possible categories, only those categories that represent 10% or more of the final result are shown. Internal factors dominate any other category by a factor of approximately 300%.

Closer inspection of points classified as internal or schedule factors reveals that many, if not most, of the entries

[1] In an attempt to reduce possible bias, entries were reviewed with minimal identifying information and categorization of the "what went right" entries was performed independently of the "what went wrong" entries.

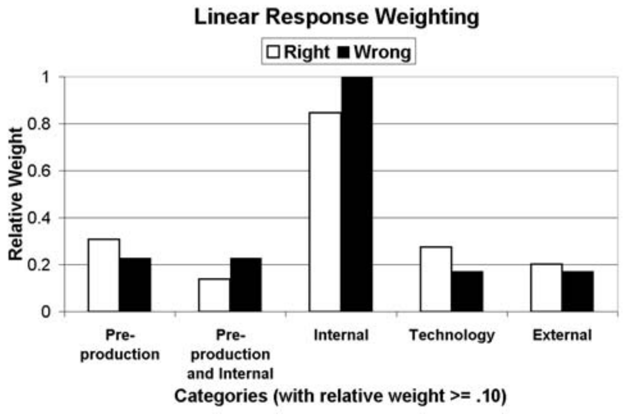


Figure 2. Observational Report Analysis

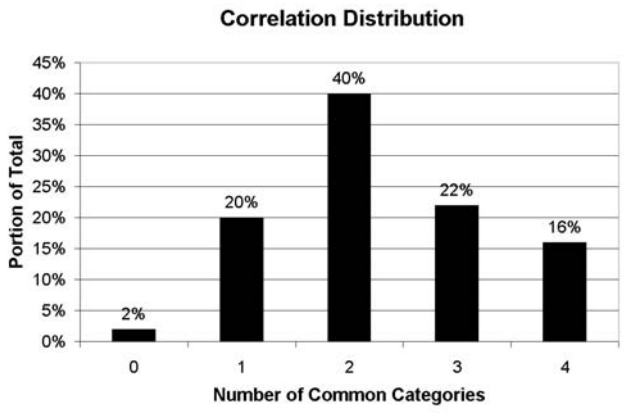


Figure 3. Correlation Within a Project

are related to classic project management issues. For example, PM4W5[2] notes “inadequate planning, PM20W3 claims a lack of success due to “underestimating the scope of tasks PM9W3 calls their schedule “too aggressive, PM18W2 states that “clear goals are great when they are realistic and PM21W1 states that “an unrealistic schedule can’t be saved without pain. It appears that these issues could be addressed by a RE process that better manages the transition from preproduction to production.

Of interest is the balance in the categorization results. Across all categories, *across all projects*, the maximum deviation from the mean is only 7.7% – a category was perceived as likely to contribute to the success of a project as it was to the failure of the project. The high degree of correlation between the “what went right and “what went wrong entries could be a result of the granularity of the categorization scheme – approximately 60% of all entries are categorized within the (major) *internal* category or related minor categories.

In general, the management of different aspects of the production process was often listed both as an element that went right and an element that went wrong within a given project. For example, in the internal category, PM3 considered their ability to focus on the task at hand as a success, while stating that “inadequate planning caused significant issues. PM21 felt that experienced personnel and internal communication contributed to success, yet stated that their “Conventions should have been better documented, communicated, and adhered to. PM27 had “strong quality assurance yet asserted that they were weak when documenting their internal standards and processes, claiming that the “Design document (was) not implemented effectively. These apparent contradictions (such as strong QA but weak internal standards) are a common theme in the observational reports.

[2]Project coding: **PM**[Project number **1..50**][**R**ight | **W**rong][Entry **1..5**]

The degree of correlation between the “what went right and “what went wrong entries *within a given project* is also significant. We assumed that the order in which the entries were presented was irrelevant and then cross-checked the results of the categorization process to see if the same categories were being reported as success and as failures. It appears (Figure 3) that individual categories are just as likely to be viewed, *within a given project*, as a contributor to success as to failure.

In an effort to determine whether these strong correlations are related to the categorization process or are inherent within the data, we are currently performing a more detailed analysis of these reports. The current analysis has identified particular challenges for requirements engineering in this domain, presented for discussion in Section 8.

For the interested reader, the postmortem columns provide further details. Domain specific successes included: PM11R1 “maintained . . . style of gameplay, PM27R2 has “ gameplay driven design, PM28R1 “created *deep* characters, PM40R1 focused on “great art. Examples of issues in preproduction included: PM2W1 “(there was a) lack of up-front design, PM6W2 “(the) game was too hard, and PM28W3 “(too much) gee-whiz factor.

6 Examples From Real Games

The initial results from our analysis of the Postmortem columns led us to conclude that weak management of the transition from preproduction to production was a source of many issues in video game development. We now look at some examples from real games that have either been published or are currently in development[3] to establish further support for this conclusion. We look at 3 issues in particular: documentation transformation, implication creating

[3]In the first example, minor changes have been made to the material to obfuscate the source.

emergent requirements, and the effects of *a priori* knowledge, to situate them within the domain and within the larger realm of requirements engineering.

6.1 Documentation Transformation

A microcosm of the documentation transformation issue is shown in Table 1. The game designer begins (1) with a story written in a narrative style. That story is then translated (elsewhere in the game design document) to a more formal form (2) that describes the action as a task and a justification for that task. The requirements engineer analyzes this information, in context (3), to determine a set of requirements: identifying in-game assets such as the player avatar, Anna (a Non-Player Character (NPC)) and an inventory item. A state that controls the players progress through the game is also identified and captured. Depending on the in-house process used, the detailed description (4) of these in-game assets may be part of the requirements document or part of a specification document. Independent of where the detailed descriptions are located, they could easily reach 50 pages once issues like artistic style, animation, and game state are included.

Performing and managing this transformation is complex. Each of these documents requires a different writing style and a single individual may not have the requisite writing skills to author materials for all purposes. In addition, creating the requirements document or specification document often requires considerable *a priori* knowledge of the available technology so that the requirements can be presented in context. There is also a multiplicative effect: each successive document is larger than the prior document as the author(s) attempt to precisely capture the required information. The authors must manage multiple stakeholder viewpoints, synthesizing a common domain language, numerous nonfunctional requirements, and inconsistencies as the project evolves.

The list of required skills is long (*e.g.* game design, requirements engineering, and technical communications) and implies a team effort. The associated costs are significant, leading to a strong management bias toward minimizing the documentation effort.

6.2 Implication

By its nature as a creative work, a game design document is replete with implied information. *Identifying* these implications requires careful analysis, *understanding* the ramifications of the implications requires significant domain knowledge.

To expand on the importance of domain knowledge, we revisit Table 1. This table captures what we call first-level implications: those implications that can be derived directly from the materials presented. Almost all development teams, independent of their experience levels, capture these implications. Missing implications at this level is usually an oversight on the part of the team.

The second level of implication requires general knowledge of the domain – in this case, the adventure game genre. These implications are generally captured by teams with members who have experience with non-trivial software development projects in the domain. In this case, the description contains significant implications regarding the game world: the characters must be situated within the appropriate environment(s). Therefore, there is an environment surrounding the player when they receive the information, there is Annas office, perhaps an office building with other office interiors, background sounds, and possibly even other NPCs in the office areas. And, if there are other NPCs, do the NPCs interact with the Player?

These second level implications could easily amount to many person-months of development effort by modelers, artists, animators, and other members of the production team.

The third level of implication requires knowledge of implementation details such as the target architecture. These implications are captured by experienced teams, particularly when the present project is a sequel of some form. The requirement for the player to visit Anna raises questions about the connectivity between the elements (locales) of the virtual world – is there more than one way the Player can get to Anna the Lawyer? How does the player experience the journey – via a scene change? Or, must they guide their avatar through the virtual world (implying the creation of all the media assets to represent the world)?

Perhaps more importantly, does the connectivity change over time? Dynamic connectivity has significant implications for representing game state (the current state of the world simulation). Designing, verifying, and maintaining a *stateful* world is more complex than a *stateless* world.

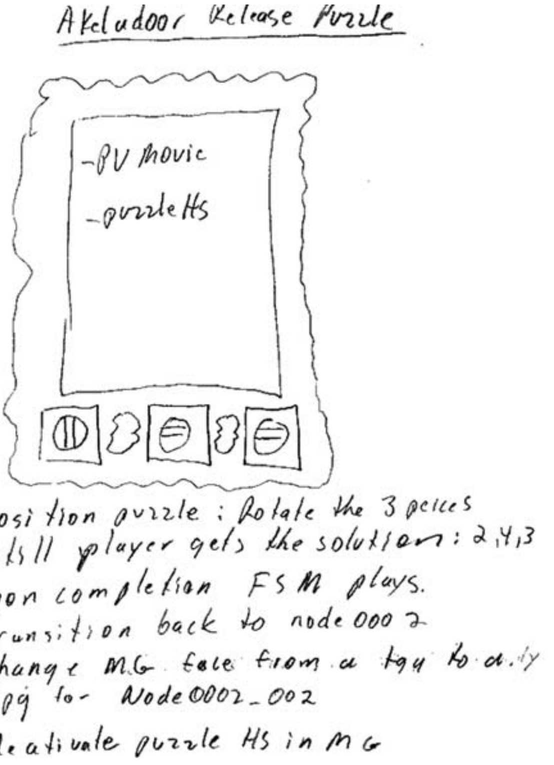
A question is raised by identifying these three levels of implication: Is it more appropriate to follow a traditional iterative process and allow these issues to surface later, or should this feedback be applied as early as possible in the process? Intuitively, early feedback is better. However, early feedback could have a negative effect on the creative process: if the game design team feels that the production team is going to reject their proposals then they may become conditioned to be less creative. The effects of early production feedback on the preproduction process merits further investigation.

Table 1. Documentation Transformation

1	Story	After her father, Bernard, died, Crystal did not know which way to turn – paralyzed by her loss until the fateful day when his Will was read.
2	Gameplay	The Player must visit Anna the Lawyer to receive a copy of Bernard's Last Will and Testament, thereby obtaining the information necessary to progress to the next goal.
3	Requirements	The Player must be represented by an avatar. Female Non Player Character required: Anna the Lawyer Inventory Item: Last Will and Testament (LWT) Player can not progress beyond Game State XYZ until LWT added to Inventory
4	Specifications	Could easily reach 50 pages

6.3 *A Priori* Knowledge

Building on the analysis of the prior section, we now look more closely at the effect of *a priori* knowledge on the requirements engineering process.



Akeladoor Release Puzzle

4 position puzzle : Rotate the 3 peices untill player gets the solution : 2,4,3
-Upon completion FSM plays.
-Transition back to node 0002
-Change MG face from a tga to a .ly jpg to- Node0002_002
- deativate puzzle HS in MG

Figure 4. Akeladoor Puzzle Description, *used with permission*

Domain specific terms, particularly abbreviations and acronyms, are common in working papers. Figure 4 is the game designers description of the *Akeladoor Release Puzzle* from the game *Apocalypse Spell*, currently under development by Far Vista Studios. Upon inspection, we see *PV Movie:* Partial Video, a less than full screen video clip, *puzzle HS:* a puzzle Hot Spot, an interaction point for the player, *FSM:* Finite State Machine, *MG:* Master Guidelines (the game uses a model driven architecture whose repository is called the *Master Guidelines* by the team).

If one attempts to formalize this document, they must understand large portions of both the preproduction and production realms. In a typical studio, this implies senior personnel from the preproduction or production staff but they are usually "too busy to perform the task. Documentation is often assigned to a junior staff member with the rationalization that this task will "bring them up to speed.

Another alternative is to add professional technical writing resources to the projects. However, there is often a perception that it takes more time to explain it to the technical writer than it does to just write it oneself. Once this excuse is in place, no writer is hired, and soon, little or no documentation is maintained.

Significant elements of the game design documentation are informal, often with substantial visual content. Visual content is particularly difficult to represent in a formal manner: iterations are often sketched as shown in the *Pyramid Puzzle* description of Figure 5. Careful examination of Figure 5 reveals evidence of prior iterations that were simply erased. Maintaining an iteration history of sketches, such as this working paper, is challenging. An electronic form of the working paper may have captured the revisions, but probably would not have captured the justifications for making the changes – often an important piece of information later in the development cycle. These justifications could lead to evolutionary changes in the game engine, perhaps even to a product family architecture.

A detailed explanation of the puzzle is beyond the scope of this paper – suffice it to say that it is a combinational puzzle that requires the player to generate the correct sequence of symbols on the screens below the pyramid, one sequence for each corner of the base of the pyramid. However, application of domain knowledge during the requirements capture phase led to significant changes in the design of the puzzle.

The first issue was puzzle complexity. Solution hints

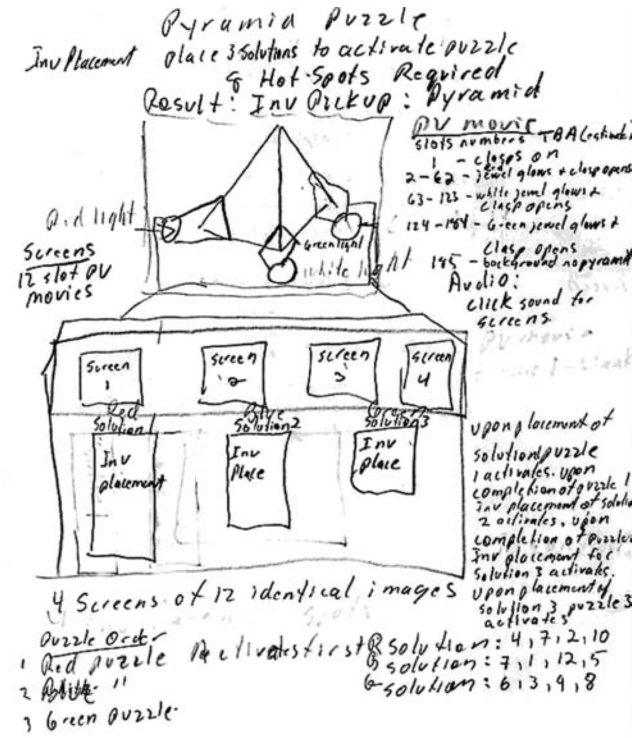

Pyramid Puzzle
Inv Placement place 3 solutions to activate puzzle
& Hot-Spots Required
Result: Inv Pickup: Pyramid
PV movie TBA (estimated)
slots numbers
1 - clasps on
2-62 - Red jewel glows + clasp opens
63-123 - white jewel glows & clasp opens
124-184 - Green jewel glows & clasp opens
185 - background no pyramid
Audio:
click sound for screens
Red light
Screens
12 slot PV movies
Green light
white light
Screen 1
Screen 2
Screen 3
Screen 4
Red Solution 1
Blue Solution 2
Green Solution 3
Inv placement
Inv Place
Inv place
upon placement of solution puzzle 1 activates. upon completion of puzzle 1 Inv placement of solution 2 activates. upon completion of puzzle Inv placement for solution 3 activates. upon placement of solution 3 puzzle 3 activates
4 Screens of 12 identical images
Puzzle Order
1 Red puzzle
2 Blue "
3 Green puzzle
Activates first
R solution: 4, 7, 2, 10
B solution: 7, 1, 12, 5
G solution: 6, 3, 9, 8

Figure 5. Pyramid Puzzle Description, *used with permission*

were provided in the form of inventory items that looked like papyrus scrolls but there was no way for the player to show the scroll and the puzzle at the same time – the game engine simply did not support simultaneous operation of inventory inspection and puzzle modes.

The game designer was informed of this restriction and it was suggested that a place be made on the puzzle for the player to "hang the scrolls so that they could see them while playing the puzzle. The result of this feedback was the layout of Figure 5 where the scrolls for each corner had a specific location (shown as *Inv Placement Blocks*).

This new layout raised an issue of screen resolution. The puzzle design called for an upper region for special effects, a middle region for puzzle input, and a lower region for puzzle solution hints. Unfortunately, this layout was beyond the resolution of the target platform so an alternative layout was required.

The final layout, shown in Figure 6, is a compromise between the game designers vision, the technical capabilities of the game engine, and the technology constraints of the target platform. Only one hints scroll is visible at a time, requiring the player to shift between inventory and puzzle modes for each corner of the pyramid – not an ideal solution from a human factors perspective, but the best that could be achieved within the constraints.

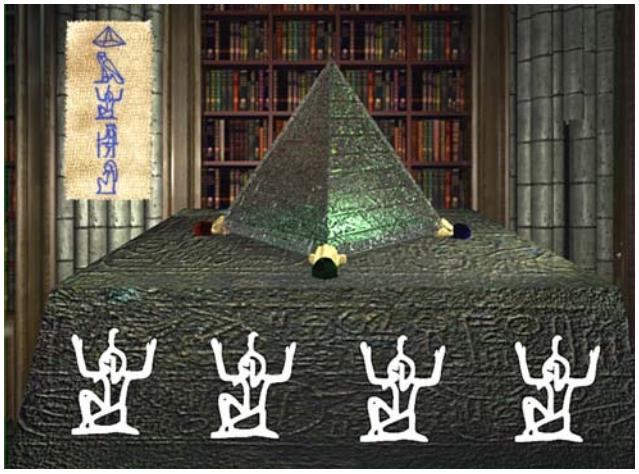


Figure 6. Pyramid puzzle prototype, *used with permission*

In this example, success was achieved through dialog between team members. Unfortunately, the revised requirements and specifications for the final product were never formally captured. Given that this is one of approximately 100 puzzles in the game, the cost of formal capture for all puzzles is significant.

The single sheet description of the puzzle resulted in the creation of the following assets: four new inventory items, 12 secondary screen elements for user interaction, three animation sequences of four seconds duration, and sound effects for user interaction and animation support. On the software side, four state machines for validating user input and three state machines for the individual corner puzzles were required. Interactions with the game world state, the current player state, inventory management, and the save game subsystem also had to be managed. None of these assets were explicitly identified by the designer; rather, they were implied in the description of the puzzle. It can be argued that identifying these implied assets if a function of the design process. However, accurately predicting the magnitude of the production effort requires their identification at the earliest possible stage in the process.

Given that this was just was one of approximately 100 puzzles in the game, it is highly desirable that the process for identifying the implied assets and side-effects be efficient. However, we are unaware of any work in this area.

6.4 Evaluation

The challenges associated with the Pyramid Puzzle are typical of the issues reported in the Postmortem columns. Using the same categories as Section 5, the terse puzzle description (assuming significant domain knowledge) is a *preproduction* issue. The puzzle description called for features that the underlying *technology* could not deliver. The technology constraints of the target platform (as defined by *external* market forces) caused a number of game design iterations. *Internal* issues, such as design complexity, design iteration, and emergent requirements interfering with test plan development, made it difficult to predict a *schedule* for this task with reasonable accuracy. The interactions are non-trivial and, when coupled with the complexities of media production, bring a unique flavor to requirements engineering in this domain.

7 Summary and Conclusions

We have analyzed the video game development process from the perspective of requirements engineering, presented a model for video game development that integrates preproduction with production, and situated the game design document as an artifact of the preproduction process. Our analysis of 50 observational reports from the Postmortem column in Game Developer magazine showed that project management issues are the greatest contributors to success or failure in video game development. In the case of failure, many of these issues can be traced back to inadequate requirements engineering during the transition from preproduction to production.

Three examples from real video games provide further evidence of the importance of properly managing the transition from preproduction to production. These examples illustrate the challenges associated with transforming preproduction documents to production documents, the importance of detecting implied information as early as possible, and the effects of applying *a priori* knowledge from the production domain to the transition from preproduction to production.

The Pyramid Puzzle example showed that, if early versions of preproduction documentation are fed forward to the production team then the production team can provide important feedback to the preproduction team. This communication cycle enables earlier identification of emergent requirements and production constraints and may improve the reliability of the transition from preproduction to production. However, the introduction of production personnel into the preproduction process may have a negative effect on the creativity of the preproduction team.

We show that requirements engineering practitioners can identify at least three levels of implication: (1) those implications that can be derived directly from the materials presented, (2) those implications that can only be derived with the introduction of general knowledge of the domain, and (3) those implications that can only be derived with the introduction of implementation details such as the target architecture. There is a strong relationship between experience and the ability to identify issues at each level of implication – indicating that a formal process for identifying implied information would not necessarily enable individuals with lesser experience to handle higher levels of implication without further guidance.

We postulate that the exploratory nature of attempts to capture the game design vision and the consequent number of production iterations is due to a lack of formal process for managing the preproduction to production transition. As project complexity increases, we predict that studios will shift to more formal processes to increase the probability of success in their development efforts despite internal resistance to this formalization.

We conclude that creating documentation to support the transition from game design document through formal requirements and specifications is difficult, requiring significant preproduction and production domain knowledge to perform successfully. A formal process to support this transition would likely increase the reliability of the process.

8 Challenges for Requirements Engineering

Is requirements engineering for video games unique? Analysis of the postmortem documents and game examples reveals that the video game industry could learn a great deal from current research and practice in requirements engineering and project management. Issues particularly notable due to their significance to game development success, and their relevance to requirements engineering, include: (1) communication between stakeholders of disparate background, (2) remaining focused on the goal and resisting feature creep, (3) influence of prior work (*e.g.*, building a new game on top of an existing game), (4) media and technology interaction and integration, (5) the importance of non-functional requirements, and (6) gameplay requirements.

Communication, focus, and prior work issues are relatively common in requirements engineering. Media and technology interaction, and the dominance of NFRs are also experienced (to a lesser extent) in other multimedia development efforts. Gameplay requirements are unique to video games.

8.1 Media and Technology

Creating a video game requires the creation of numerous software artifacts. Not only must the game engine be developed but a media production pipeline is also required. The pipeline must be designed and the tools associated with the pipeline must be built while keeping in mind that these are tools for artists and animators as well as for technical personnel.

Technology requirements often emerge as media assets are integrated into the game engine. The actual player experience delivered by the game engine may not meet the requirements of the game designer and publishers. Minimum platform targets may change due to technological advances and marketplace pressures it is not uncommon to have to rework media assets developed early in a project to make them appear less dated by the end of the project.

Requirements engineering for media production in video game development is particularly challenging due to the interactions between the requirements of the video game artifact, the requirements of the tools needed to create the video game artifact, and the strongly differentiated user groups.

8.2 The Importance of NFRs

Video games are designed to entertain. Therefore, non-functional requirements such as *fun*, storyline, continuity, aesthetics, and flow must dominate their requirements specification. However, there are no established practices for capturing and specifying such NFRs – requirements engineering can make a significant contribution in this area.

Validation of gaming NFRs is very complex. Generally, an abstract NFR like *fun* is highly dependent on the target market something that is fun for a young child may be annoying to an adult. The link between NFRs and target markets or user demographics has not yet been explored by RE in this domain.

Verification of gaming NFRs, and functional requirements related to media assets, is also complex. Requirement verification via test is particularly difficult when the requirement is to engender emotions in the user.

8.3 Gameplay

It is usually through gameplay that NFRs like *fun* and flow are achieved. One can argue that it is the NFRs and gameplay that make each video game unique. For example, the dominant video game genre is the first person shooter, made famous by the *Doom* and *Quake* series from id Software. All first person shooter video games share a common set of core technologies required by that genre: protagonist avatar(s), antagonist(s), the ability to move the protagonist avatar within the virtual world in an acceptably realistic manner, and the ability for the protagonist avatar to choose and use a weapon to wreak mayhem upon the antagonists. It is the presentation of these core technologies to the user (via gameplay, storyline, and aesthetic elements such as art and sound) that makes each game unique.

Storyboards in video game development are more closely related to storyboards in animated movie production (evaluating aesthetics and storyline) than the typical user-interaction scenario development in productivity application software development. Storyboards are also used by some developers as a first step in prototyping gameplay – a means for assessing the player experience.

Prototyping gameplay is particularly challenging. It is difficult to assess the player experience early in the development cycle for significant progress must be made on building the underlying game engine infrastructure before gameplay testing can begin. This is a particularly high-risk scenario due to the likelihood that new requirements will emerge as gameplay testing continues, new requirements that must be tracked, and for which test plans must be developed. The emerging requirements may even force significant changes to the fundamental architecture of the system that, in extreme cases, may cause project failure.

9 Future Work

We are currently performing a more detailed analysis of observational reports from Game Developer magazine and other sources. We expect this information to further guide the development of a process for managing the transition between preproduction and production. Mechanisms for capturing and stating non-functional requirements, such as *fun*, in a manner that can be validated, measured, and verified are also required.

Involving production personnel in the preproduction process may lead to more efficient development or it may lead to reduced creativity. Further investigation is needed to quantify the tradeoffs.

10 Acknowledgements

We wish to thank Game Developer magazine for continuing the *Postmortem* column over the years. The first author thanks Electronic Arts for the opportunity to present an early version of this (and related) work at their British Columbia studios and for the valuable feedback he received. We also thank Richard Buckley of Far Vista Studios for access to internal game design documentation prior to release of their game.

References

[1] A. I. Anton. Goal-based requirements analysis. In *ICRE '96: Proceedings of the 2nd International Conference on Requirements Engineering (ICRE '96)*, page 136, Washington, DC, USA, 1996. IEEE Computer Society.

[2] C. E. Association. *Digital America*. Published electronically at http://www.ce.org, 2003.

[3] V. Authors. Postmortem column. *Game Developer*, 6(5) through 11(6), May 1999 - June 2004.

[4] T. Bentley, L. Johnston, and K. von Baggo. Putting some emotion into requirements engineering. In *Proceedings of the 7th Australian Workshop on Requirements Engineering*, 2002.

[5] E. Bethke. *Game Development and Production*. Wordware Publishing, Inc., 2003.

[6] B. Boehm and V. Basili. Software defect reduction top 10 list. *IEEE Computer*, 34(1):135–137, Jan. 2001.

[7] K. Breitman and J. C. S. do Prado Leite. Ontology as a requirements engineering product. In *Requirements Engineering*, pages 309–319, 2003.

[8] L. Chung. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[9] M. Csikszenthmihalyi. *Flow: The Psychology of Optimal Experience*. Harper Perennial, 1990.

[10] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. In *6IWSSD: Selected Papers of the Sixth International Workshop on Software Specification and Design*, pages 3–50, Amsterdam, The Netherlands, The Netherlands, 1993. Elsevier Science Publishers B. V.

[11] S. W. Draper. Analysing fun as a candidate software requirement. *Personal Technology*, 3(1):1–6, 1999.

[12] A. G. Editor. *POSTMORTEMS from Game Developer*. CMP Books, 2003.

[13] B. Fuson. *2003 Top Boxoffice*. Published electronically at http://www.hollywoodreporter.com, 2003.

[14] J. A. Goguen. The dry and the wet. In *ISCO*, pages 1–17, 1992.

[15] J. A. Goguen and C. Linde. Techniques for requirements elicination. In *Proceedings of the International Symposium on Requirements Engineering*, pages 152–164, Los Alamitos, California, 1993. IEEE CS Press.

[16] I. H. Holbrook. A scenario-based methodology for conducting requirements elicitation. *SIGSOFT Softw. Eng. Notes*, 15(1):95–104, 1990.

[17] M. Hassenzahl, A. Beu, and M. Burmester. Engineering joy. *IEEE Software*, 18(1):70–76, 2001.

[18] M. Jarke, K. Pohl, R. Doemges, S. Jacobs, and H. Nissen. Requirements information management: The NATURE approach. *Ingenerie des Systemes d'Informations*, 2(6):609–637, 1994.

[19] F. D. Laramee, editor. *Game Design Perspectives*. Charles River Media, Inc., 2002.

[20] A. Marcus. The emotion commotion. *interactions*, 10(6):28–34, 2003.

[21] N. Medvidovic and D. S. Rosenblum. Domains of concern in software architectures and architecture description languages. In *Proceedings of the 1997 USENIX Conference on Domain-Specific Languages*, 1997.

[22] D. Michael. *The Indie Game Development Survival Guide*. Charles River Media, Inc., 2003.

[23] D. A. Norman. *The Design of Everyday Things*. Doubleday Books by permission of Basic Books, 1988.

[24] J. N. och Dag, V. Gervasi, S. Brinkkemper, and B. Regnell. Speeding up requirements management in a product software company: Linking customer wishes to product requirements through linguistic engineering. In *RE*, pages 283–294, 2004.

[25] J. Robertson. Point/counterpoint: Requirements analysts must also be inventors. *IEEE Software*, 22(1):48–51, Jan. 2005.

[26] S. Robertson. *Requirements Trawling: techniques for discovering requirements*. Published electronically at http://www.systemsguild.com/GuildSite/Robs/trawling.html, 2004.

[27] A. Rollings and D. Morris. *Game Architecture and Design, A New Edition*. New Riders Publishing, 2004.

[28] K. Salen and E. Zimmerman. *Rules of Play: Game Design Fundamentals*. MIT Press, 2004.

[29] M. Saltzzman, editor. *Game Design Secrets of the Sages*. Macmillan Publishing USA, 2000.

[30] A. van Lamsweerde. Requirements engineering in the year 00: a research perspective. In *International Conference on Software Engineering*, pages 5–19, 2000.

[31] P. Zave. Classification of research efforts in requirements engineering. *ACM Computing Surveys*, 29(4):315–321, 1997.