# What Went Wrong? A Survey of Problems in Game Development

FÁBIO PETRILLO, MARCELO PIMENTA, FRANCISCO TRINDADE, and CARLOS DIETRICH
Institute of Informatics, Federal University of Rio Grande do Sul, Brazil

Despite its growth and profitability, many reports about game projects show that their production is not a simple task, but one beset by common problems and still distant from having a healthy and synergetic work process. The goal of this article is to survey the problems in the development process of electronic games, which are mainly collected from game postmortems, by exploring their similarities and differences to well-known problems in traditional information systems.

Categories and Subject Descriptors: K.8.0 [**Personal Computing**]: General—*Games*; D.2.9 [**Software Engineering**]: Management; K.6.3 [**Management of Computing and Information Systems**]: Software Management

General Terms: Management, Human Factors

Additional Key Words and Phrases: Electronic games, game development, problems in game development, survey, postmortems

**ACM Reference Format:**

Petrillo, F., Pimenta, M., Trindade, F., and Dietrich, C. 2009. What went wrong? A survey of problems in game development. ACM Comput. Entertain. 7, 1, Article 13 (February 2009), 22 pages. DOI = 10.1145/1486508.1486521 http://doi.acm.org/10.1145/1486508.1486521

## 1. INTRODUCTION

The game development industry is entering a new age in which technology and creativity go together, producing some of the most astonishing entertainment activities of the 21st century. This industry has an annual income of billions of dollars; in 2003 its revenues exceeded those of the movie industry. It is one of the most powerful, exciting, and influential industries in the arts [Gershenfeld et al. 2003]. Great game projects count on highly specialized multidisciplinary teams that need simultaneously software developers, designers, musicians, scriptwriters, and many other professionals. Thus, the career of

Authors' address: Institute of Informatics, Federal University of Rio Grande do Sul, Brazil; contact author's email address: Fabio@petrillo.com

game developer is currently one of most dynamic, creative, challenging, and potentially lucrative.

However, game development is an extremely complex activity, and a much harder task than one can initially imagine. Creating a game is a different experience from what it was in 1994, and certainly much more difficult, having grown in complexity in recent years [Blow 2004].

The main goal of this work is to collect and discuss the problems that afflict the electronic game industry, since surprisingly very little attention has been paid to collecting information about them, to oganizing this information in a systematic form, and to quantifying the frequency with which they occur.

To achieve our goal we have taken the following steps:

- analyzed the problems of the traditional software industry, collecting this information to compare it with the game industry;
- analyzed the problems of the game industry that are cited in specialized literature, making it possible to map such problems when analyzing the postmortems;
- collected via postmortem analysis (focusing on aspects of software engineering), the real problems cited by professionals in the game industry and made a qualitative analysis of the outcome;
- compared the results obtained from the game industry to the data from the traditional industry and searched for similarities and idiosyncrasies.

This article is structured as follows: Section 2 presents a study of the problems of the traditional software industry, providing statistics on project performance and discussing problems in software engineering. In Section 3 we discuss the problems cited in the specialized literature on game development and in the postmortems; we number them and quote examples that demonstrate their relevance to the project. After this we discuss the methodology used in the analysis of postmortems and describe the results obtained in the problems survey and compare the results to the problems discussed in Section 2. Finally, in Section 4 we conclude and propose future work.

## 2. PROBLEMS IN SOFTWARE INDUSTRY

The analysis of problems in the development of software systems goes back to the origins of software engineering. The famous NATO conference in 1968 [NATO 1968], in which the term software enginnering was created, had as its basic goal the analysis of the key problems in the critical areas of program construction.

Although the academic community has being discussing the problems that afflict the development of computational systems for almost 40 years, we still have difficulties in elaborating elegant and correct software that satisfies customers and meets estimates, as much in time as in budget. The current status of software development can be seen as far from ideal. Jones [1995] suggests that software is one of the most problematic aspects of the corporate world: budgets and deadlines are not met and errors are more common in large software projects than in almost any other business area. Projects fail in all countries, in big and small companies, in commercial and governmental organizations, regardless of status or reputation.

Data provided by the Standish Group [1995] shows that more than 30% of projects are cancelled before being completed and that 53% exceed their budgets by 189%. Only 16% of projects are completed in the agreed period ot time and budget with all the initially specified functionalities. In this scenario, Sommerville [2001] suggests that software engineering is not in crisis, but in a state of chronic agony.

Due to such difficulties many software projects fail, but these failures are not reflected in change. As cited in The Chaos Report [Standish Group 1995], many errors are not investigated in depth and so the same errors continue to be made. The corporate insanity in software development is shown by the fact that the same kinds of mistakes are repeated in different contexts, yet different results are expected every time. For these reasons, DeMarco and Lister [2003] contend that the main problems of software projects are not technological but managerial.

The Standish Group quantified the main factors that cause damage to or cancellations of projects. In a questionnaire, IT executives were asked what the main factors that caused problems in projects were: 13.1% of the executives cited incomplete requirements, and 12.4% the lack of user envolvement. In Section 3.5, we will use some of these results to compare the traditional software industry and the electronic games industry.

From these surveys and from an analysis of the references made by Yourdon [2003] and Charette [2005], it is possible to group the problems into four main categories: (1) scheduling problems; (2) budget problems; (3) quality problems; and (4) management- and business-related problems. In fact, IT projects rarely fail for just one or two reasons alone, more frequently there is a combination of interrelated problems. In the following sections, we will analyze each one of them, characterize them, and discuss their causes.

## 2.1 Scheduling Problems

All project that do not meet their deadlines have scheduling problems. As cited by Brooks [1995], more than all other causes put together, the biggest one for cancelling a project is scheduling.

Perhaps the main reason for poor scheduling is the weakness of the techniques used to estimate the time to completion, which confuse effort with progress. With problematic estimates, progress is poorly monitored and when delays are endountered, the natural answer is to add more professional help to the project.

In an article about the limits of defining estimates in software projects, Lewis [2001] comments that many programmers themselves discover that it is impossible to make accurate estimates for software projects.

Another factor that leads to scheduling problems is optimism. Brooks [1995] claims that all programmers are optimistic. Such optimism, along with the naivety frequently associated with inexperience, means people have no realistic idea of how much time and effort will be necessary to construct a system. This situation is so common that DeMarco and Lister [2003] call it “hysterical optimism“: everbody in the organization desperately wants to deliver a complex system that could realistically only be completed in at least three years in nine months.

Cowardice is another sociological aspect found while estimating projects. Brooks [1995] observes that programmers and their bosses are influenced by the sponsor’s demands during the elaboration of the project schedule. These demands can even decide the time a task should take to complete, but they cannot decide the time necessary for its accomplishment. False schedules are produced to meet the sponsor’s desires for speed, and this behavior is much more common in software projects than in any other engineering area.

Scheduling problems due to overly optimistic estimates, cowardice, and uncertainties have damaged the credibility of the software industry. Lewis [2001] claims that credibility will not be regained while absolute deadlines continue to be promised. The software industry must acquire the knowledge about the existing uncertainties and intrinsic risks of software development that is necessary to promote a public and honest discussion about such risks.

### 2.2 Budget Problems

Projects with budget problems are characterized by demands for investments above the estimates. Models to estimate the cost of a project are based on a function of the necessary development time or on measures of size, counted as lines of code or as points of the function [Lewis 2001]. Consequently, the cost of a product is also a function of the number of professionals needed for its development [Brooks 1995], as persons per month or persons per hour.

On average, the budgets of software projects exceed by almost 200% the original estimated costs, a phenomenon that occurs in companies of all sizes [Standish Group 1995]. The Standish Group study shows that some projects (4.4%) exceed their budgets by more than 400%.

### 2.3 Quality Problems

From a customer’s point of view, a project has quality problems when the final product is different from the one expected. Quality is associated with the customer’s satisfaction, aesthetic aspects, and the system’s ablity to meet its requirements [Bach 1995].

From the technical point of view, the most common reason for disasters in software projects is poor quality control. According to Jones, finding and correcting bugs are the most time consuming and expensive aspects of software development, especially for large systems.

According to the Standish Group [1995], of all projects with problems, one quarter was delivered with 25% to 49% of the originally specified functionalities. On average, 61% of the originally specified functionalities were present in the end product.

### 2.4 Management Problems

Management problems happen in several ways, including bad communication, lack of investment in the team’s training, and neglecting to inspect the project at regular intervals [Charette 2005]. According to Jones [1995], bad pproject management is the origin of many problems in the process of software development.

Decisions are frequently difficult to make because they involve choices based on incomplete knowledge, which turns estimating how much a project will cost or how much time it will take into more of an art than a science [Charette 2005].

## 3. PROBLEMS IN THE ELECTRONIC GAMES INDUSTRY

In this section we discuss the problems of the electronic games industry. We will initially analyze what postmortems are and show the problems cited in the specialized literature on game development, demonstrating its characteristics and pointing out the criteria used to define the problems. Finally, we will compare the results with the data obtained from the traditional software industry.

### 3.1 The Game Postmortems

The term *postmortem* designates a document that summarizes the project development experience, with a strong emphasis on the positive and negative aspects of the development cycle [Hamann 2003]. It is usually done by managers or senior project participants right after a project finishes [Callele 2005]. From a software engineering viewpoint, postmortems are important tools for managing knowledge [Birk et al. 2002] from which the group can learn from its own experiences and plan future projects. In fact, the postmortem analysis can be so revealing that some authors [Birk et al. 2002] argue that no project should be finished without a postmortem.

Postmortems are much used in the game industry. Many game websites devote entire sections to these documents (e.g., *Gamasutra* [http://www.gamasutra.com] and *Gamedev* [http://www.gamedev.net]). It is also very interesting to note the variety of development team profiles and projects behind these documents, varying from few developers in small projects to dozens of developers in five-year-long projects.

The postmortems published by *Gamasutra* mainly follow the structure proposed by the *Open Letter Template* [Myllyaho et al. 2004], which is composed of three sections. The first section summarizes the project and presents some important aspects of the development; the next two sections, however, discuss the aspects most interesting to game developers:

*What went right*: A discussion of the *best practices* adopted by developers, solutions, improvements, and project management decisions that improved the efficiency of the team. All these aspects are critical elements for planning future projects.

*What went wrong*: A discussion of the difficulties, pitfalls, and mistakes experienced by the development team, in both the technical and management aspects.

The postmortem is closed with a final message from the author, commonly followed by a project technical brief, which includes the number of full- and part-time developers, length of development time, release date, target platforms, and the hardware and software used in the development.

The information contained in the postmortems constitutes a knowledge base that can be reused by any development team; it includes examples and real-life development experiences. It can help in knowledge sharing, and be very useful for planning future projects.

## 3.2 Problems in Specialized Literature

According to Flood [2003], all game development postmortems say the same things: the project was delivered behind schedule; it contained many defects; the functionalities were not the ones that had originally been projected; it took a lot of pressure and an immense number of development hours to complete the project.

The specialized literature on electronic games' development is also good at describing the problems of this industry. For Bethke [2003], the electronic games industry generally adopts a poor methodology (if any), causing projects to last longer than they were suposed to, going over budget, and full of defects.

Such a scenario is responsible for declarations like this:

> "It's a rare project where there is not some form of agony. I am trying to think of one that wasn't miserable in some way. I can think of one that was very smooth, but in general, I think you'll find that there is always some degree og angst associated with making game". Ian Lane, Mad Doc Software

and

> "Every projects has a moment where you are totally in hell, especially when you are making new intellectual property or trying to create the "fun" and core game identity". Scott Campbell, president and founder, Incognito Studio

In order to elucidate these problems better, we will discuss some of them at great length, based on the causes for the mistakes referred to by Flynt and Salem [2004] and by studying the pecialized literaature on the development of electronic games.

***Problems of Scope.*** According to Flynt and Salem [2004], the biggest reason for mistakes in game projects is the failure to clearly establish the project's scope. If a project does not have a well-established target, the requirements that emerge can cause significant structural changes to the system's architecture, and cause serious problems

The development teams get lost when when the scope of the project grows very large and complex, and they have to deal with the highly specific requirements of the games domain as well [Gershenfeld et al. 2003; Blow 2004]. However, the main cause of problems is the commonly occuring situation where new functionalities are added during the development phase, thus increasing the project's size. This practice is known in the industry as feature creep.

Features are added late when the developer, managers, or customers add requirements to the project after its scope has already been defined. The intention is to create the best possible game, so more elements are added, perhaps resulting in a more interesting game (with no guarantees) but certainly in a bigger one.

This occurs for many reasons. According to Flynt and Salem [2004], the most common one is when, by chance, programmers discover new and interesting features during the development process and decide to add them to the game or when they randomly add functionalities they consider attractive. In defense, Flynt and Salem argue that this occurs because the development phase is initiated without a strong effort at analysis being made, in addition to the lack of techniques for improving the investigation and verification of project requirements.

Another common form of feature creep happens when external code (a new component, for example) is incorporated without planning so as to save time. However, it frequently becomes necessary to work extremely hard to integrate this new component. A third form of feature creep takes place when developers decide, despite having a set of solid libraries, to implement their own algorithms.

However, therre are a lot of examples in the games industry where features discovered during the development phase transformed a game into a success. Game development is not a linear process [Flynt and Salem 2004]. Hence, if an interesting function is discovered, it must be analyzed in terms of risk and, if viable, be added to the project's schedule.

***Scheduling Problems.*** The specialized literature on electronic games describes in exhaustive detail the scheduing problems in game projects. Although they usually begin with reasonably structured schedules, bright ideas, and enthusiasm, something always goes wrong. And the number of things that can go wrong is virtually endless [Bethke 2003]. These problems are often associated with the many disciplines needed to develop game projects, which generates delays caused by waiting for the work of others, as well the lack of a realistic estimate in the initial plan, making the team incapable of fixing a deadline for the project [Flynt and Salem 2004].

Estimating the time needed to complete a task entails the risk of underestimates, causing cumulative scheduling delays. According to Flynt and Salem [2004], developers recurrently fail in their estimates due to a lack of historical data that would assist in estimating the time needed to finish a task. Another common problem in estimating deadlines is not taking the time spent in meetings and other such activities into account, besides not planning for time to correct defects in the game.

However, for Flynt and Salem [2004] the key to the scheduling problem is the imbalance among quality, cost, and opportunity. If a game is delivered two months after the release of a similar game from another company, the possibilities for the success of the game released later are diminished. On the other hand, a game with a significant number of defects that is launched early can experience more problems than if it were delivered later with fewer bugs.

***Crunch Time.*** Crunch time is a basketball term that describes the last minutes of the gameduring which both teams fight for victory.

In the games industry, the term is used for periods of extreme work overload, typically in the last weeks before the validation phase, and mainly in the weeks that precede the final deadline for project delivery. In these periods, a work load of more than 12 hours a day is common, from 6 to 7 days per week, without intervals for rest.

Although traditional software companies undergo such cycles of work overload, Gershenfeld et al. [2003] claims that the electronic games industry goes through periods of crunch time more frequently. This assertion is confirmed by Hamann's [2003] description of his experience:

> "On one game I worked on, one of the team members complained to the producer that months of working 12-hours-per-day, seven-days-a-week was getting to him. He said he'd like to have at least one weekend off. The producer replied, 'This is the game industry—crunch time is a fact of life'. I have a question for all of you game company owners, producers and leads out there: Ever wonder why you have such a high turnover rate at your company? Studies have shown that most game developers will put up with a severe crunch mode for one, or at most two, projects and then they start looking for another company. Believing that crunch time is a fact of life is a result of following the blind religion that's so pervasive in our industry".

Gershenfeld et al. [2003] comment that periods of crunch time can be good for single or ambitious people, who make their work the main focus of their lives. However, they claim that this situation is not sustainable for people who have a family or want to maintain a serious relationship. Hence they suggest constructing schedules that plan for a healthy pace of daily work, proposing that people are much more efficient if they work a typical eight-hour wor day.

***Technological Problems.*** All games depend on technology. Moreover, the technologies used in games are so advanced that game companies are the leaders in graphical computation. However, cutting-edge technology brings risks as well, since using it may frequently involve great effort and a large investment in time, as described by Flynt and Salem [2004].

According to Gershenfeld et al. [2003], the technological risks are generally higher when the team is working on a new platform that has not been completely delivered or consolidated. The first of these risks is that no developer has ever worked in it; the second is that frequently the platform hardware still contains problems that are only found by development team at the *launch title.*[1]

## 3.3 Problems in Postmortems

Although the specialized literature on the development of games contains valuable descriptions concerning the problems of this industry, as we saw in Section 3.2, another way to look at the problems is by reading *postmortems*. This reading is deeper (more forceful and specific) and can accomplish more than reading the specialized literature.

Twenty postmortems were published at the *Gamasutra* site with the goal of examining and analyzing the problems of the electronic games industry. The analysis was done through the reading of stories in detail, extracting the ideas that demonstrate the problems faced by the teams during the game development process.

[1]*Launch title* is a game that is delivered at the same time that the new platform.

We will describe the main problems found by the postmortems, quoting examples to define the analytic criteria for the survey described in Section 3.4. We analyzed 15 problems that are the most relevant and recur most frequently in the postmortems.

***Unrealistic Scope.*** It is not surprising that postmortems explicitly report that many projects are unrealistic or extremely ambitious in scope. The *Gabriel Knight 3* project [Bilas 2000], for example, contains the following sentence: "[it] was an extremely ambitious project combined with an extremely inexperienced team".

Some projects like *Command and Conquer: Tiberian Sun* [Stojsavljevic 2000], *Vampire: The Masquerade* [Huebner 2000], and *The X-Files* [VandenBerghe 1999] dedicate a complete section to reporting on this problem. Huebner comments that "Many of the team members had wanted for some time to do a really huge, ambitious role-playing game".

Stojsavljevic describes "unrealistic expectations" and that "The degree of hype and expectations that *Tiberian Sun* had to fulfill was staggering". Moreover, writing on the *Black & White* project, Molyneux [2001] states that "looking back, I don't know whether we were insanely ambitious, because at the time we started, you couldn't have done what we did".

***Feature Creep.*** Feature creep is the process by which new modules are added without planning during software construction. This problem is described regarding the *Vampire* project [Huebner 2000]:

> "New engine features get added too late in the schedule to be utilized fully by the designers and artists. This happened several times during Vampire. Some of the more interesting special effects, for example, were added only a few weeks before the data was to be locked down for final testing".

This could also be seen clearly in the comments made by Reinhart [2000] about the *Unreal Tournament* project:

> "The amount of content grew and we soon realized we had a much larger project on our hands than we had originally thought. . . . *Unreal Tournament* is a very fun game with a lot of features packed into a short amount of development time. Those features were largely added through spur-of-the-moment decisions".

Phrases such as these are found in many postmortems: "As we realized this late in the project . . . " [Smith 2001]; "humans procrastinate. . . . Audio was plugged in at the last moment" [Scheib 2001]; or "we were capable of incorporating several features that originally had not been planned to be developed".

Fristrom [2000] reported his experience with this problem during the *Draconus* project:

> "Another example of poor planning was that magic was an afterthought. We always knew we would have spells in the game, special abilities with nice special effects that your character would acquire as the game progressed, but they were left until late in development because there were always more pressing issues. Of course, once we did implement spells, they unbalanced the play of the game horrendously, and much more unplanned work was needed to make magic function reasonably".

***Cutting Features During Development.*** As well as feature creep there is another problem that is mentioned frequently: that of cutting features during the development process. Due to the fact that projects initiate an overly ambitious number of functionalities, some of which are even implemented, but for a variety of reasons they end up being cut further on in the project.

A typical case is found in *Rangers Lead the Way* [Ridgway 2000], in which the initial features of the game were modified to maintain the schedule and the budget:

> "We dropped the Macintosh version right away, and we terminated the PlayStation version at alpha (when it was still running at five to eight FPS). Networking support was postponed for an expansion pack a year into development".

Or as reported by Fristrom [2000]:

> "Our answer to being under funded and under scheduled was to cut network play. We needed to make much bigger cuts".

Another example of this situation is due to an unrealistic scope. The report made by Malenfant [2000[ for the game *Wild 9* describes this well:

> "The material they came up with is enough to fill up three or four games like this one. In addition, the whole team took this game to heart and ended up coming up with suggestions of their own. We were thus faced with a situation toward the end of the project: we could not fit everything into the game".

For *Tropico* [Smith 2001], this problem was due to a different reason: lack of physical space necesssary to cut features. Smith comments that in this situation "we had to cut other features to create space, features which would have improved the game".

***Problems in the Design Phase.*** Although designing is a common practice in game projects, mainly by using a design document or game bible, many reports claim that there are problems in this phase. The postmortem on *Wild 9* brings emblematic testimony:

> "Many times, titles are late because certain gaps in the original design were overlooked and the full design was never really laid down on paper before development started".

Barrett et al. [2002] dedicate an entire section called "Design on the fly" ro this problem:

> "Because we were designing a game to the technology (rather than the other way around), we were throwing out design documents as quickly as they could be written. Art assets had to be revised, retextured, discarded, and rebuilt from scratch several times. As most readers will know from experience, this is a scenario for feature creep, obsolete tool sets, and blown deadlines".

On the other hand, Saladino [1999] says that the opposite situation may also be a problem:

> "One interesting example of the opposite problem arose in our custom interface system: we had too much design. Our interface system is the library of code that handles our front end interface and our in-game heads up display. The code

> was written by a programmer with extensive experience in C++ object oriented design and loved to use it. He spent many weeks designing and crafting an interface system which was extremely . . . well . . . designed. Private data fields were placed into private headers. The ”const” operator was used everywhere. Void pointers hiding internal data structures made debugging a nightmare”.

Moreover, he makes the following recommendation:

> “There are extremes at both ends of the design spectrum and either one can seriously damage overall productivity. Make sure that you spend time designing what is important”.

***Delays.*** Many game projects mention delayed schedules. For example, Ridgway [2000] gives a clear description of this problem in the *Rangers Lead the Way* projetct: “We learned the hard way what’s possible to accomplish in 15 months; as a result, we completed the game in 20”.

Ragaini [2002] analyzes this problem at great depth and describes its main cause:

> “Depending on how far back you look at the schedules, *Asheron’s Call* was either one to two years late. . . . Deadlines were consistently missed. A lot of this was due simply to underestimating the time required for development tasks. This created a domino effect as we continually played catch-up, trying desperately to make up for lost time”.

VandenBerghe [1999] describes this problem likewise: “The most severe blow suffered by all teams was from accepting an unrealistic schedule”. Again, optimism was the cause of the problem; also demonstrated by Fristrom’s [2000] statement regarding the *Draconus* project: “In my experience, publishers love overoptimistic schedules”.

***Technological Problems.*** The failures of third-party application programming interfaces (APIs) for the platform or hardware were classified as technological problems. A typical example appeared in the *Age of Empires II: The Age of Kings* project [Pritchard 2000]:

> “Microsoft’s DirectPlay API still has a number of issues that make it less than perfect. One of its biggest problems is documentation and testing of the lesser-used portions of the API, or rather the lack thereof. Late in the development of AoK, our communications programmer, Paul Bettner, was able to communicate with the DirectPlay developers and an interesting scenario played out several times: Paul would attempt to solve some problem and the developers would indicate that it wouldn’t work because of bugs in DirectPlay that they knew about but that were not documented”.

There were also more prosaic problems as, for instance, in the *Draconus* project [Fristrom 2000] with the C compiler: “We lost many weeks in our battle to find a compiler that could actually compile our code”.

In the end, the use of external components from third parties frequently produces more difficulties in the development process. A typical example is reported by Upton [2000]:

> “Our external solutions for rendering and networking both fell through and had to be replaced with internally developed code late in the development

> cycle. . . . In retrospect, we would have saved money and had a much smoother development process if we'd bitten the bullet early on and committed ourselves to building our own technology base".

***Crunch Time.*** As discussed previously, crunch time was found in many postmortems. Perhaps the most eloquent testimony about this problem was made by VandenBerghe abaout the *The X-Files* project:

> "The most severe blow suffered by all teams was from accepting an unrealistic schedule. Despite endemic problems, . . . the concept that was floated at the time was that it would be possible to adhere to the original schedule if everyone simply worked around the clock. Foolish and naïve, we bought it, and started pushing. While a final push of a few weeks or even a few months is common toward the end of a technical project, the schedule of six to seven days a week of twelve- to twenty-hour days adopted by The X-Files team stretched on for eight months. Predictably, exhaustion began degrading the ability of people to produce quality work, not to mention the deeply straining effects it had personally on team members and their families. In retrospect, the death march did little to speed the project's finish. There were several systems that were developed during this time that caused us no end of grief in development, largely because of the inability of the developers to make clear and rational design decisions from lack of sleep. . . . For me, the lesson here is that if the schedule begins to slip, the solution is not as simple as just applying more work to the problem. Pushing harder won't necessarily finish the project any sooner, and some of the damage done to the team under that much stress may be irreparable".

Sleep deprivation was also cited by Spanel and Spanel [2001] writing about the *Operation Flashpoint* project:

> "After a long, sleepless night of playing through the game and fixing any problems that appeared, everything looked fine, and most of the team could finally go to sleep again".

Moreover, experience from the *Diablo* [Schaefer 2000] project shows that past experiences do not necessarily produce best practices for new projects:

> "The original *Diablo* went gold on the day after Christmas in 1996, after a grueling four-month crunch period. We hadn't put any thought into what game to do next, but as most developers can probably relate to, we were pretty certain we weren't ready to return to the Diablo world after such a long development cycle. The only thing we were certain of was that we wanted to avoid another crunch like we had just experienced. *Diablo II* went gold on June 15, 2000, after a grueling 12-month crunch period".

***Lack of Documentation.*** Although many projects reported sucess with documentation processes, mainly with the project document, many postmortems cite congenital problems due to lack of documentation, as described by Spanel and Spanel [2001] for *Operation Flashpoint*:

> "Lack of documentation is a common affliction among game developers, but some aspects of this problem were so severe in our case that they are worth mentioning. While we'd never believed too much in designing the game on paper, the real problem was that we never even had documentation of the things that we'd finished. This situation led to incredible problems in the final stages of development. Many tasks could only be done by one person on the whole team. In other cases, hours were spent trying to investigate how something

> had originally been meant to work. We recognized these problems and tried to improve them, but apart from a few instances, our effort wasn't really successful. As the development team grew, the missing documentation was becoming a more serious problem. But the final project deadlines were getting closer as well, so it was nearly impossible to find the time to address the problem".

Even though Reinhart [2000] did not have a project document, he still argues for its utility:

> "If we develop a design document, we'll use it with the understanding that it can be modified at any time. That having been said, I think there is a definite positive argument for having some sort of central guide to everyone's ideas. Having the ability to sit down and look over the big picture is very valuable".

***Communication Problems.*** Such problems are also described in postmortems, manly between tecnical and art teams, as reported by Fristrom [2000] regarding the *Draconus* project: "The game designers and artists didn't really communicate as well as they should have on the levels they made".

Another interesting example was the report about communication problems between the tecnical team and the publisher during quality control [Ragaini 2002]:

> "Communication between Microsoft and Turbine was also a major factor. The teams were separated by about 3,000 miles and three time zones. Although weekly conference calls were scheduled, they lacked the collaborative mentality necessary for maintaining a successful relationship. E-mail threads were either ignored or else escalated into tense phone calls, and in some cases the bug-tracking database (RAID) was not used effectively. Clearly, everyone would have benefited from more face-to-face time. E-mail—and even conference calls - are poor media for managing new and sensitive corporate relationships, especially ones between companies with such different corporate cultures. From a developer's perspective, it's always easy to blame the publisher for unrealistic expectations and bureaucracy. What's important to realize is that it is everyone's obligation to communicate expectations and problems before they escalate to the point of being a crisis".

***Tool Problems.*** When elaborating a game, tools become essential elements in the construction process, for project control, for generating a version for delivery, and for elaborating parts of a game.

The *Diablo II* project had problems with its tools [Schaefer 2000]:

> "The greatest deficiency of our tools was that they did not operate within our game engine. We should have made tools that let us create content within the game engine".

Fristrom [2000] gives a dramatic description of how the lack of effective tools can be problematic: "The process for making a release build was insane".

For some projects, the tool problem was related to version control, as described by VandenBerghe [1999] re *The X-Files*:

> "We used SourceSafe 4.0 for our source control. It worked effectively as a code database, but was a less effective choice for a revision control system. It gave us one nasty shock during the PSX port: we lost all of our revision history due to being unable to move the revision database from one server volume to another (larger) one, but fortunately this didn't end up hurting the project much".

***Test Problems.*** Some projects had problems with testing. One of the most explicit criticisms was made by Bernstein [2002] in a postmortem:

> "If we had done more beta testing, with a larger group and earlier on, we would have gotten the kind of outside feedback that would have helped us realize that some of the tradeoffs we were making were going the wrong way"

Upton [2000] said the following about the game *Rainbow Six*:

> "We got lucky. As a result of our early missteps, the only way we could get the game done on time was to cut deeply into our testing schedule. We were still finding new crash bugs a week before gold master; if any of these had required major reengineering to fix, we would have been in deep trouble."

Insufficient testing was identified by Meynink [2000] as a problem associated with publishers as well:

> "Never underestimate your testing requirements and, more importantly, never let your publisher underestimate your testing requirements. Almost obvious to anyone who has developed a game should be the testing phase, especially for projects with shorter deadlines. Make sure that the level of testing support you expect is written into the contract with your publisher. . . . In the future, we will be sure to have a testing plan explicitly stated in both the contract and the development schedule".

***Team Bulding.*** Some postmortems describe problems in team building, problems that caused communication and relationship difficulties. *Wild* 9 is an example where the project had two different teams [Malenfant 2000]:

> "The first *Wild* 9 team was a great collection of talented individuals, while the second was a talented team. This statement alone sums up the main problem encountered by the first team. As we found out later in the project, communication and good atmosphere were two factors that made this project move forward; it's now obvious that the best thing to do was to start over rather than try to salvage a situation that was going nowhere".

Another unusual situation was described by Ridgway [2000] regarding *Rangers Lead the Way*:

> "The last thing that I expected was that it would be hard to find good programmers in Seattle. I was hired at the start of SpecOps' development and didn't manage to bring the entire programming staff on board for nearly seven months. Needless to say, this caused substantial delays. We had similar problems hiring the art team".

Experience is an essential element in game development. Ragaini [2002] describes the make-up of his team and the effects of his choices:

> "None of the senior developers at Turbine (including me) had ever shipped a retail PC game. None. Many of the employees were students immediately out of college, or even college students completing a work-study program. This obviously was the source of several severe problems in the development of *Asheron's Call*".

***Number of Defects.*** Some projects were characterized by the great number of defects found in a development phase; a comment about the *Draconus* project makes this explicit: "The bug list went over three thousand" [Fristrom 2000].

However, the most marked testimony about this problem was given by Molyneux regarding the *Black & White* game:

> "After canceling our Christmas party on December 26, 2000, we managed to hit Alpha, which as any developer knows is a very loose definition, but at least we could say that all the game features were now locked. After a well-deserved Christmas break, we came back to find that we had more than 3,000 bugs. We had six weeks to reduce this to zero, but the thing about bug-fixing is that you can solve one problem but in doing so create three more. So although we worked as hard as we could, the overall figure crept down slowly rather than dropped at the rate at which we were actually sorting out the bugs. . . . The irony was that the last 10 bugs were the hardest to fix, and with every one there were four more created. It was as if the game just didn't want to be finished and perfected".

***Loss of Provessionals.*** Some commented that their projects were seriously affected by the loss of professionals; this was confirmed by Upton [2000] in his comments on the *Rainbow Six* project:

> "Losing even a junior member of a development team close to gold master can be devastating. When our lead engineer took ill in February 1998, we were faced with a serious crisis".

This problem also afflicted the *Rangers Lead the Way* project; but in this case the problem was in the art team [Ridgway 2000]:

> "About a month before E3 '97 rolled around, our art lead and a senior artist decided to leave the company. I had designed most of Viper's capabilities with them, and losing them at that critical time really devastated the project".

***Over Budget.*** This problem was the least cited one, being that only two projects explicitly referred to it: *"about $400,000 more than we has budgeted to tackle it"* [Fristrom 2000].

However, it was the *Gabriel Knight* 3 project that brought this problem to the fore, since the original estimate was considerably less than US$ 2 million, but the final budget came to US$ 4.2 million [Bilas 2000].

## 3.4 Analysis of Postmortems and Research Results

The electronic games industry for reasons of competitiveness and its corporate style, generally does not make its internal data regarding projects accessible to researchers [Callele et al. 2005]. To avoid this difficulty, the postmortem analysis described in Section 3.1 was used instead.

The 20 postmortems analyzed in this survey are listed in Table I. From the table, one can see that projects of various team sizes, budgets, and development times were analyzed. It is also important to point out that all the postmortems we analyzed were finished projects; that is, projects that resulted in complete games that were delivered to the market.

The survey process took place in three stages. In the first stage, each postmortem was read and interesting citations ware highlighted; in the second, the problems were classified for tabulation using the survey in Section 3.2; and in the third stage, each postmortem was read again, with special attention paid

Table I. Postmortems Analyzed

| Project | Team | Development Time (Months) |
|---|---|---|
| Beam Runner Hyper Cross | 5 | 4 |
| Dark Age of Camelot | 25 | 18 |
| Black & Whire | 25 | 37 |
| Rangers Lead the Way | 13 | 20 |
| Wild 9 | 18 | NI |
| Trade Empires | 9 | 15 |
| Rainbow Six | 22 | 24 |
| The X-Files | 36 | 48 |
| Draconus | 19 | 10 |
| Cel Damage | 16 | 24 |
| Comand and Conquer: Tiberian Sun | 35 | 36 |
| Asheron's Call Massive Multiplayer | 60 | 48 |
| Age of Empires II: The Age of Kings | 40 | 24 |
| Diablo II | 40 | 36 |
| Operation Flashpoint | 10 | 48 |
| Hidden Evil | 22 | 12 |
| Resident Evil 2 | 9 | 12 |
| Vampire: The Masquerade | 12 | 24 |
| Unreal Tournament | 16 | 18 |
| Tropico | 10 | 12 |

to the highlighted sentences, tabulating each sentence according to the classification made previously; the final result produced Table II.

In Table II, the columns are reserved for problems and the games are arranged in lines. Each postmortem was studied for each one of the 15 problems discussed in Section 3.3. Each problem was marked with a "Yes". This value was only attributed to the problems mentioned explicitly by the author, as one can see in the examples described in Section 3.3. Problems not mentioned by the author, or cited explicitly or that did not occur were marked with a "No". It is important to mention that when this value is given to a certain problem it does not necessarily mean that the problem did not occur, but that it may have been considered irrelevant by the postmortem's author.

The number of occurrences of "Yes" was counted in lines and columns so that we could quantify the problems found in the postmortems,. The number of "Yes"-es in the lines represent how many different types of problems a certain game presented. On the other hand, the number counted in the columns represents the number of occurrences of this problem in the set of analyzed games. We can see in the penultimate line that the table was organized in a decreasing sequence of occurrences. The last line contains the percentage of occurrences in relation to the number of projects studied.

When we analyze this results more closely, we see that the most cited problems are the *unreal or ambitious scope* and *features creep* 75% (15 out of 20) of projects reporting this problems. After that, demonstrating the coherence of the postmortems, 70% of projects cited the *cutting features during development process.* The other most frequently found (65%) were *problems in the design phase* and *delay or optimistic schedule*. *Technological problems* (60%) can also

Table II. Occurrence of Problems in the Projetcs

| Game | Unreal or ambitious scope | Feature creep | Cutting features | Design problems | Delay or optimistic schedule | Technological problems | Crunch Time | Lack of Documentation | Communication problems | Tools problems | Test Problems | Team building problems | Great number of defects | Loss of Professionals | Over Budget | Total | % |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Beam Runner Hyper Cross | No | Yes | Yes | Yes | No | No | Yes | Yes | No | No | No | No | Yes | No | Yes | 7 | 46,7% |
| Gabriel Knights | Yes | Yes | Yes | No | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | 13 | 86,7% |
| Black & Whire | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | No | Yes | No | No | Yes | No | No | 9 | 60,0% |
| Rangers Lead the Way | Yes | Yes | Yes | Yes | Yes | No | No | No | No | No | No | Yes | No | Yes | No | 7 | 46,7% |
| Wild 9 | Yes | Yes | Yes | No | No | Yes | No | Yes | Yes | No | No | Yes | No | Yes | No | 8 | 53,3% |
| Trade Empires | No | Yes | Yes | Yes | No | No | No | No | No | No | Yes | No | No | No | No | 4 | 26,7% |
| Rainbow Six | Yes | Yes | No | Yes | Yes | Yes | Yes | Yes | No | No | Yes | Yes | Yes | Yes | No | 11 | 73,3% |
| The X-Files | Yes | No | No | Yes | Yes | Yes | Yes | No | No | Yes | No | No | No | No | No | 6 | 40,0% |
| Draconus | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | Yes | No | Yes | No | Yes | 12 | 80,0% |
| Cel Damage | No | Yes | Yes | Yes | No | No | No | No | No | No | No | No | No | No | No | 3 | 20,0% |
| Comand and Conquer: Tiberian Sun | Yes | Yes | Yes | No | Yes | Yes | No | No | No | No | No | No | No | No | No | 5 | 33,3% |
| Asheron's Call | Yes | Yes | Yes | No | Yes | No | No | Yes | No | No | Yes | Yes | No | No | No | 7 | 46,7% |
| Age of Empires II: The Age of Kings | Yes | No | No | No | Yes | Yes | Yes | No | Yes | No | No | No | No | No | No | 5 | 33,3% |
| Diablo II | Yes | No | No | No | Yes | Yes | Yes | No | No | Yes | No | No | No | No | No | 5 | 33,3% |
| Operation Flashpoint | Yes | Yes | Yes | No | Yes | Yes | Yes | Yes | Yes | No | Yes | Yes | No | Yes | Yes | 12 | 80,0% |
| Hidden Evil | Yes | No | Yes | Yes | No | No | No | No | Yes | Yes | No | No | No | No | No | 5 | 33,3% |
| Resident Evil 2 | Yes | No | No | Yes | Yes | Yes | No | No | No | Yes | Yes | No | No | No | No | 6 | 40,0% |
| Vampire: The Masquerade | Yes | Yes | Yes | Yes | Yes | Yes | No | No | No | No | No | No | Yes | No | No | 7 | 46,7% |
| Unreal Tournament | No | Yes | No | Yes | No | No | No | Yes | Yes | Yes | No | Yes | No | No | Yes | 7 | 46,7% |
| Tropico | No | Yes | Yes | Yes | No | No | No | Yes | No | No | No | No | No | No | No | 4 | 26,7% |
| Occurrences | 15 | 15 | 14 | 13 | 13 | 12 | 9 | 8 | 7 | 7 | 7 | 7 | 6 | 5 | 5 | **143** | **7,2** |
| % | 75% | 75% | 70% | 65% | 65% | 60% | 45% | 40% | 35% | 35% | 35% | 35% | 30% | 25% | 25% | **47,7%** | **47,7%** |

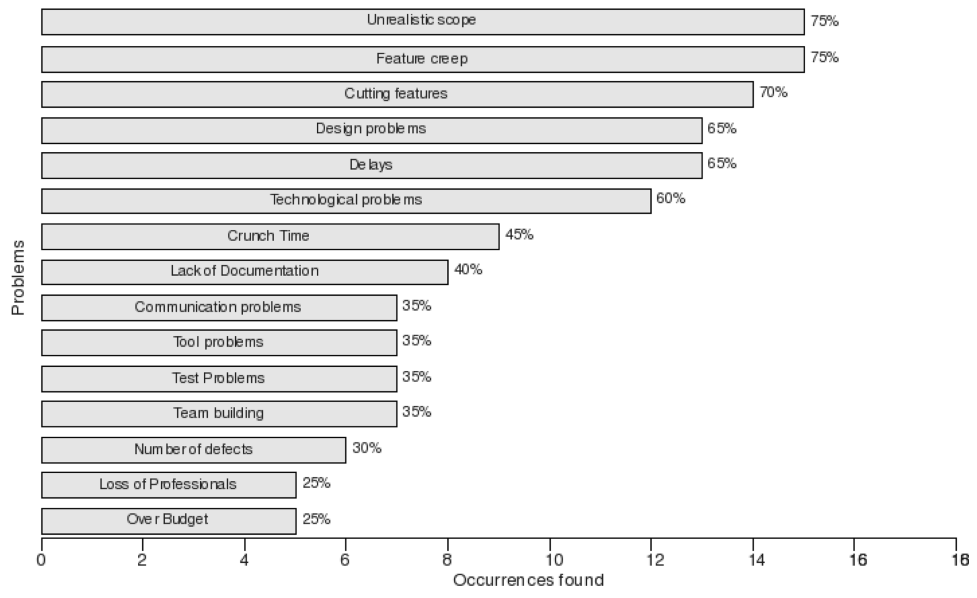

Fig. 1. Occurrence of problems.

be highlighted. Figure 1 shows the histogram of the occurrences of problems in decreasing sequence, so that we can see a comparison of the results in a graph.

The most surprising result is that problems such as *crunch time* and *over budget*, said to be "universal," occurred at a relatively low rate; crunch time was a problem in only 45% of the projects. The *over budget* and *loss of professionals* problems had the lowest incidence among all the analyzed problems, that is, only 25%.

## 3.5 Comparing the Traditional and Electronic Games Industries

Finally, to develop games is to develop software [Bethke 2003; Gershenfeld et al. 2003]. The statement also suggests that it is possible to compare, into terms of problems, game development with the traditional software industry. In order to elaborate this comparison, we use the problem survey in Section 2 and the results of research in Sections 3.2 and 3.4.

When we first compare these two studies, we see that in fact *all the main problems of the traditional software industry are also found in the eletronic games industry.* In both studies, the unrealistic scope is highlighted, as well as the problems with the requirements analysis. Charette [2005] also points out that the unrealistic objectives and undefined requirements are among the main problems, strengthening the similarities between the two industries.

But of all the problems, the one that is almost as universal in information systems as in games projects is that of unrealilistic scope or exaggerated optimism. With a percentage of 75% of projects in analyzed games, and as clearly highlighted by Yourdon [2003], Brooks [1995], and DeMarco and Lister [2003], it is not surprising that optimism and naivety in defining the scope as well as in estimating the effort needed to do a task, are the factors that deterimine the occurrence of most problems in the two industries. Moreover, both industries suffer from the Machiavellian attitude of senior executives and game publishers. For these reasons, neither the traditional software industry nor the games industry suffers from technological problems, but from management problems.

In terms of differences between the two industries, an important problem, specific to the games industry, is that of communication among teams. The team in traditional software engineering is usually relatively homogeneous, made up of technicians in information technology. However, the electronic games industry, because it is multidisciplinary, attracts people with a variety of profiles such as plastic artists, musicians, scriptwriters, and software engineers. This mixture of people, even though it provides the more creative work environment, seems to produce a split in the team, dividing it into “the artists” and “the programmers”. This division, which does not really exist in the traditional software industry, causes important communication problems [Flynt and Salem 2004], since both teams believe in communicating clearly but use their specific vocabularies to express their ideas. This is an important source of misunderstanding.

Another important diference is that elaborating game requirements is much more complex, since efficient methods to determine subjective elements such as “fun” do not exist. The point of Callele et al. [2005] is to clearly state that it is necessary to extend the traditional techniques of requirements engineering to support the creative process in the development of electronic games.

## 4. CONCLUSIONS AND FUTURE WORK

From the research done with the postmortems analysis on aspects of software engineering, we see that the most cited problems are the *unrealistic or ambitious scope* and *features creep* with 75% (15 out of 20) of the projects complaining about these two problems. This was followed by the *cutting of features during the development phase*, with 70% of the projects citing it; and in third place, the *problems in the design phase* and *delays or too opstimistic project schedules*, with 65%. The technological problems, with 60%, may also be cited.

The study of postmortems shows that the *unreal scope or the exagerated optimism*, mainly in estimating how much effort a project will take, are probably important factors for the occurrence of the majority of problems. This situation can be seen as being almost universal, since it is found as much in information systems as in game projects, with a 75% percentage of ocurrences in the analyzed game projects, clearly pointed out by Yourdon [2003], Brooks [1995], and DeMarco and Lister [2003]. From these results, it is possible to conclude that the traditional and game software industries do not suffer from technological problems, but from managerial ones.

In fact, *all the main problems of the traditional software industry are also found in the games industry,* hence we can conclude that they are related. In both studies, unrealistic scope was the problem that needed requirements definition.

The so-called universal problems of the games industry such as *crunch time* and *over budget* had a relatively *low* occurrence rate, 45% for crunch time.

Problems with *over budget* and the *loss of professionals* had the lowest incidence among all the problems analyzed, at a rate of only 25%.

Comparing the two industries, we can perceive similarities in the frequency of some problems (e.g., schedule delays and requirement problems having almost equal rates). There was a considerable discrepancy regarding budget problems, probably due to the greater emphasis on the technological and management aspects (given by the stories' authors), thus minimizing the financial aspects of the project.

The main limitations we found were as follows:

- Although relevant, all analyzed postmortems were created after the pprojects were concluded, with at least a game delivered. Cancelled projects were not analyzed, nor were projects that, despite effort, did not produce a complete game for the market. This may have led to optimistic results in comparison with what reality shows.
- The fact that postmortems do not have a formal structure for explanation, since they are basically nade up of stories in everyday language, the analysis of problems depended significantly on how the text was interpreted by the researchers. Thus, different criteria for judgment could be adopted by different analyzers. To try to minimize this problem, a set of criteria was used as a basis for the analysis.

The positive results obtained in this work, as well as its limitations, create some exciting possibilities for future work:

- Since this work focused on problems, the same study can be repeated to find the best sofware development practices and apply them to the games industry. Maybe we can be surprised again, but this time with the number of good practices that are adopted.
- It would be interesting future work to examine the development of specific techniques to improve the verification of requirements in electronic games projects. as suggested by Flynt and Salem [2004].
- To consider a software enginnering methodology that specializes in the domain of the games industry, and covers aspects such as complexity, multidisciplinarity, and creativity, as reommended by Callele et al. [2005].

## REFERENCES

BACH, J. 1995. The challenge of "good enough" software. *American Program. Mag.* (Oct.).

BARRETT, K., HARLEY, J., HILMER, R., POSNER, D., SNYDER, G., AND WU, D. 2002. Postmortem: Cel damage. *Gamasutra* (Feb.). http://www.gamasutra.com/features/20020227/wu_01.htm.

BERNSTEIN, R. 2002. Postmortem: Trade empires. *Gamasutra* http://www.gamasutra.com/features/20020125/bernstein_01.htm.

BETHKE, E. 2003. *Game Development and Production*. Wordware Publishing, Plano.

BILAS, S. 2000. Postmortem: Gabriel Knight 3. *Gamasutra* (Oct.). http://www.gamasutra.com/features/20001011/bilas_01.htm.

BIRK, A., DINGSOYR, T., AND STALHANE, T. 2002. Postmortem: Never leave a project without it. *IEEE Software 19*, 3 (May), 43–45.

BLOW, J. 2004. Game development: Harder than you think. *ACM Queue 1*, 10 (Feb.), 28–37.

Brooks, F. P. 1995. *The Mythical Man-Month—Essays on Software Engineering*. Addison-Wesley, Reading, ,MA.

Callele, D., Neufeld, E., and Schneider, K. 2005. Requirements engineering and the creative process in the video game industry. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*.

Charette, R. N. 2005. Why software fails. *IEEE Spectrum 42*, 9 (Sept.), 42–49.

DeMarco, T., and Lister, T. 2003. *Waltzing with Bears—Managing Risk on Software Projetcs*. Dorset House Publishing.

Flood, K. 2003. Game unified process. GameDev.Net, May. http://www.gamedev.net/reference/articles/article1940.asp.

Flynt, J. P. and Salem, O. 2004. *Software Engineering for Game Developers* (first ed.) Software Engineering Series. Course Technology PTR.

Fristrom, J. 2000. Postmortem: Draconus. *Gamasutra* (Aug.). http://www.gamasutra.com/20000814/fristrom_01.htm.

Gershenfeld, A., Loparco, M., and Barajas, C. 2003. *Game Plan: The Insider's Guide to Breaking in and Succeeding in the Computer and Vieo Game Business*. St. Martin' s Griffin Press, New York.

Hamann, W. 2003. Goodbye postmortems, hello critical stage analysis. *Gamasutra* (July). http://www.gamasutra.com/resource_guide/20030714/hamann_pfv.htm.

Huebner, R. 2000. Postmortem of nihilistic software's vampire: The masquerade – Redemption. *Gamasutra* (Aug.). http://www.gamasutra.com/features/20000802/huebner_01.htm.

Jones, C. 1995. Patterns of large software systems: Failure and success. *IEEE Computer 28*, 3 (March), 86–87.

Lewis, J. P. 2001. Limits to software estimation. *ACM SIGSOFT Software Engineering Notes 26*, 4 (July), 54–59.

Malenfant, D. 2000. Postmortem: Wild 9. *Gamasutra* (Jan.). http://www.gamasutra.com/features/20000107/wild9_01.htm.

Meynink, T. 2000. Postmortem: Resident evil 2 for the n64. *Gamasutra* (July). http://www.gamasutra.com/features/200000728/meynink_01.htm.

Molyneux, P. 2001. Postmortem: Black & white. *Gamasutra* (June). http://www.gamasutra.com/features/20010613/molyneux_01.htm.

Myllyaho, M., Salo, O., Kääriäinen, J., Hyysalo, J., and Koskela, J. 2004. A review of small and large postmortem analysis methods. In *Proceedings of the IEEE 17th International Conference on Software and Systems Engineering and their Applications.*

NATO. 1968. Software engineering—Report on a conference sponsored by the NATO science commitee. Tech. Rep., Nato Science Commitee, Garmisch, Germany, Oct.

Pressman, R. S. 2006. *Engenharia de Software* (6th ed.), McGraw-Hill, São Paulo.

Pritchard, M. 2000. Postmortem: Age of empires ii: The age of kings. *Gamasutra* (March). http://www.gamasutra.com/features/20000307/pritchard_01.htm.

Ragaini, T. 2002. Postmortem: Asheron's call. *Gamasutra*. (May). http://www.gamasutra.com/features/20000525/ragaini_01.htm.

Reinhart, B. 2000. Postmortem: Unreal tournament. *Gamasutra* (June). http://www.gamasutra.com/features/20000609/reinhart_01.htm.

Ridgway, W. 2000. Postmortem: Rangers lead the way. *Gamasutra* (Feb.). http://www.gamasutra.com/features/20000201/ridgeway_01.htm.

Saladino, M. 1999. Postmortem: Star trek: Hidden evil. *Gamasutra* (Nov.). http://www.gamasutra.com/features/19991119/startrekpostmortem_01.htm.

Schaefer, E. 2000. Postmortem: Diablo ii. *Gamasutra* (Oct.). http://www.gamasutra.com/features/20001025/schaefer_01.htm.

Scheib, V. 2001. Postmortem: Beam runner hyper cross. *Gamasutra* (Nov.). http://www.gamasutra.com/features/20011109/whoopas_01.htm.

Smith, B. 2001. Postmortem: Tropico. *Gamasutra* (Oct.). http://www.gamasutra.com/features/20011010/smith_01.htm.

Sommerville, I. 2001. *Software Engineering* (6th ed.) Addison-Wesley, London.

Spanel, O. and Spanel, M. 2001. Postmortem: Operation flashpoint. *Gamasutra* (Dec.). http://www.gamasutra.com/features/20011219/spanel_01.htm.

STANDISH GROUP. 1995. Chaos. Tech. Rep., Standish Group.

STOJSAVLJEVIC, R. 2000. Postmortem: Westwood studios' command and conquer: Tiberian sun. *Gamasutra* (April). http://www.gamasutra.com/features/20000404/tiberiansun_01.htm

UPTON, B. 2000. Postmortem: Rainbow six. *Gamasutra*. (Jan.). http://www.gamasutra.com/features/200000121/upton_01.htm.

VANDENBERGHE, J. 1999. Postmortem: The x-files. *Gamasutra* (Dec.). http://www.gamasutra.com/features/19991203/xfiles_postmortem_01.htm.

YOURDON, E. 2003. *Death March* (2nd ed.). Prentice-Hall Englewood Cliffs, NJ.

Received November 2007; revised November 2008; accepted November 2008